Universidad Autónoma de Madrid

Escuela politécnica superior

Trabajo de fin de grado

DESARROLLO DE UN ESQUEMA DE AUTENTICACIÓN BASADO EN LA IDENTIDAD PARA CLIENTES ANDROID

Juan Manuel Donaire Felipe

Tutor: David Arroyo Guardeño

Mayo 2016

Índice general

Índice de Figuras

Agradecimientos

Quería agradecer a todas aquellas personas que me han ayudado a acabar la carrera y a finalizar este proyecto.
A David, mi tutor, por ayudarme y guiarme durante estos meses.
A todos mis compañeros de clase, por demostrar año tras año que trabajando juntos se aprende más y por estar siempre dispuestos a ayudar. Sin vosotros, todas esas horas de prácticas no habrían sido lo mismo.
A todos los profesores que me han ayudado a lo largo de estos cuatro años.
A mi familia, por hacer esto posible y por apoyarme siempre.

Resumen

Con el gran crecimiento de las Tecnologías de la Información y de la Comunicación (TIC), la criptografía ocupa una posición de vital importancia para asegurar la seguridad en las comunicaciones. Uno de los desafíos de la criptografía actual se encuentra en la gestión de identidades digitales, problema cuya solución más empleada suele ser la Public Key Infrastructure (PKI) o Infraestructura de Clave Pública, en la que una Entidad certificadora emite certificados para las claves públicas de los usuarios registrados. Sin embargo, esta solución presenta unos inconvenientes que se podrían solucionar con la llamada Identity-Based Cryptography (IBC) o Criptografía Basada en la Identidad.

En este TFG se presenta un breve repaso sobre las distintas soluciones al problema de la gestión de identidades centrándose en la Criptografía Basada en la Identidad, además de discutir sus beneficios y exponer la descripción formal del proyecto software realizado para este trabajo. Dicho proyecto lleva a cabo un Cliente de mensajería instantánea para dispositivos móviles Android que se conecta con un Servidor desarrollado en Flask para Python y que hace las veces de Servidor de aplicaciones y de Private Key Generator (PKG), implementando un esquema de autenticación basado en la identidad. Para la implementación de la Seguridad Basada en la Identidad se ha utilizado la biblioteca Charm de Python.

La descripción detallada del proyecto incluye el análisis, diseño, implementación y plan de pruebas llevados a cabo para su desarrollo. Dicha descripción, además, viene acompañada de una serie de anexos con información complementaria, como la planificación temporal del proyecto o la creación de certificados SSL.

Palabras clave
Criptografía, identidad física, identidad digital, distribución de claves, RSA, SSL, PBC, IBE, PKG, Android, Charm.

Abstract

As a consequence of the leading role of Information and Communication Technologies (ICT) in today's econcomic activity, cryptography has become of the utmost importance to ensure the security of online communication. One of the biggest challenges of current cryptography lies in digital identity management. Public Key Infrastructure (PKI) is the most commonly used to handle this problem. In PKI, the Certification Authority (CA) issues certificates for the public keys of registered users. However, this solution has a few problems which could be solved by using Identity-Based Cryptography (IBC).

This document gives a brief review over the different ways of solving the digital identity management problem, focusing on Identity-Based Cryptography and its advantages. It also presents the formal description of the developed software project. This project implements an instant messaging app for an Android Client which connects with a Server developed in the Flask microframework for Python. The Server is made by different components, and it works as Private Key Generator (PKG) and as an Application Server. All in all, it implements an authentication schema based in Identity-Based Cryptography using Charm library for Python for the IBE functionality.

The detailed description of the project includes its analysis, design, implementation and test plan. It also includes some complementary information in the form of annexes like the planification of the project and the guide for the creation of SSL certificates.

Key words
Cryptography, digital identity, physical identity, key distribution, RSA, SSL, PBC, IBE, PKG, Android, Charm.

Glosario

actividad Elemento básico de una aplicación Android. Una actividad define la parte gráfica de una "pantalla" determinada, y describe también la parte lógica encargada de responder adecuadamente a los estímulos del usuario. 19, 20, 30, 31

AES Advanced Encryption Standard es uno de los algoritmos de cifrado asimétrico más populares, es el estándar de cifrado empleado por el gobierno de Estados Unidos. Emplea cifrado por bloques. 4, 25, 28, 29, 34

Android Sistema Operativo para dispositivos móviles predominante actualmente. Producto de la compañía Google. 1, 9, 10, 12, 13, 15, 18, 20, 21, 23, 24, 30, 39, 40

Charm Biblioteca criptográfica para Python que implementa las funcionalidades de la Encriptación Basada en la Identidad. 1, 14, 15, 17, 20, 21, 23, 24, 29, 39, 40

Flask Biblioteca criptográfica para Python que implementa las funcionalidades de la Encriptación Basada en la Identidad. 1, 14, 21, 37

Hash Función unidireccional que transformar un conjunto de bits de cualquier tamaño en una cadena de tamaño fijo.. 5, 6, 14, 15, 18, 24, 26, 27, 33–35

jPBC Biblioteca criptográfica para Java que implementa las funcionalidades de la Encriptación Basada en la Identidad. 14, 15

JSON JavaScript Object Notation (JSON), es un formato de texto presentado como alternativa al XML para el intercambio de datos. 24, 25, 27, 28, 30, 32, 35, 36

POST Tipo de petición HTTP donde los parámetros van en el cuerpo del mensaje y no en la cabecera. 21, 32

RSA RSA, Rivest, Shamir y Adleman es el algoritmo de encriptación asimétrica más utilizado. 4

sal Sal, o salt en inglés, es un término utilizado en criptografía para una cadena de bits aleatoria que se usa para derivar contraseñas a otras cadenas nuevas, aportando ciertas ventajas, como por ejemplo, dificultar los ataques por diccionario. 25, 27, 29, 33, 35

Spring Framework para el desarrollo de aplicaciones en Java que facilita, entre otras cosas la creacion de servidores. 14

SQL Injection SQL Injection, o inyección de código SQL es un método de aprovecharse de debilidades de sistemas con bases de datos incluyendo código SQL en ciertos campos de entrada para diversos propósitos que van desde poder acceder al sistema sin pertenecer a él, como destruir por completo la base de datos. 29

Stateless Stateless se dice de un Servicio o Servidor que no guarda sesiones. Es decir, que cada petición al servidor es única e independiente del resto y no tiene en cuenta otras peticiones anteriores. 26

Volley Volley es una librería HTTP para Android que facilita y agiliza la comunicación a través de dicho protocolo. 31, 32

Glosario de acrónimos

API Aplication Programming Interface. 12, 21, 23

CSR Certificate Signing Request. 49

HTML Hyer Text Markup Language. 37, 38

HTTP Hyer Text Transfer Protocol. 13, 14, 17, 21, 31

IBC Identity-Based Cryptography. 3, 6–8, 25, 39

IBE Identity Based Encryption. 1, 10, 14, 15, 17, 18, 23–26, 28, 29, 39

PKG Private Key Generator. 7–9, 13, 14, 17, 18, 25–27, 34, 38, 39, 45

PKI Public Key Infrastructure. 3, 6–8

REST Representation State Transfer. 13, 21

SDK Software Development Kit. 24

SSL Socket Secure Layer. 2, 15, 17, 21, 24, 32, 49

TFG Trabajo de Fin de Grado. 1, 2, 39

TIC Tecnologías de la Información y de la Comunicación. 1, 39

TLS Transport Layer Security. 21

Capítulo 1

Introducción

Vivimos en una época en la que la forma de comunicarnos ha cambiado sustancialmente en un periodo de tiempo muy corto [1]. Internet nos permite llevar a cabo los mismos actos de comunicación que la humanidad lleva realizando miles de años pero ignorando la distancia entre las personas. Sin embargo, pese al dramático cambio de condiciones, seguimos esperando que se apliquen las mismas normas, y esperamos de la nueva forma de comunicación los mismos requisitos que de la comunicación antigua [2]. Es decir, pese a no estar viendo a la persona, queremos seguridad de que estamos hablando con dicha persona. Pese a estar introduciendo información en un medio inseguro al que puede acceder el resto de la población, queremos que sólo el destinatario pueda acceder al mensaje. Y por supuesto, queremos que el mensaje no se pierda, y que llegue tal y como lo mandamos, como si esa persona estuviese delante de nosotros y estuviésemos hablando en una habitación en la que sólo estuviésemos los dos.

No obstante, cómo ya se ha dicho antes, Internet no es un medio seguro, por lo que debemos emplear herramientas que nos permitan cumplir esos requisitos [3]. Esa herramienta se llama criptografía, y la utilizamos constantemente cuando empleamos un dispositivo tecnológico para comunicarnos con otros. Sin embargo, no existen propuestas perfectas que den solución a los problemas anteriormente citados, especialmente para el problema de gestión de identidades.

1.1. Objetivos y motivaciones

Los objetivos y motivaciones principales de este TFG son tanto académicos, como técnicos y personales.

Desde la parte académica, aunque no es el foco principal de este trabajo, se pretende realizar un estudio sobre el problema de la gestión de identidades en la criptografía. Hacer un repaso de las tecnologías actuales, como los anillos de confianza y la Infraestructura de Clave Pública [4], centrándose en la Criptografía Basada en la Identidad. Este tipo de criptografía es una tecnología muy nueva y aún poco utilizada que proporciona soluciones sencillas a problemas complejos con la tecnología empleada actualmente, como es la revocación de certificados [5].

La parte técnica, que forma el grueso del trabajo, consiste en implementar un producto software de mensajería para móviles Android, construyendo también la parte del Servidor. En el Servidor, entre otros, se implementa la parte criptográfica correspondiente a IBE [6] empleando la biblioteca Charm de Python [7]. El uso de esta biblioteca constituye un reto, pues está aún en desarrollo y presenta una clara limitación tanto en documentación como en funcionalidad.

Por último, en lo personal, el objetivo principal es aprender acerca del problema de gestión de identidades, así como de la Criptografía Basada en la Identidad [8]. Por otro lado, la consideración de un caso de uso concreto ha permitido familiarizarse con una serie de tecnologías de gran potencial en el actual contexto TIC [9], más específicamente en los ámbitos de Cloud y el Internet of Things. Android, IBE a través Charm, Flask [10] y Python son algunos ejemplos de dichas tecnologías. Finalmente, y no falto de importancia, desarrollar por primera vez un producto completo, diseñándolo desde el principio con la seguridad como un requisito sustancial. Esto es, se ha pretendido desarrollar el proyecto aplicando el criterio *security-by-design* [11].

1.2. Estructura del documento

El presente documento está organizado en siete capítulos.

El primer capítulo es la introducción, que resume el proyecto y expone sus principales objetivos, así como las motivaciones a la hora de realizarlo.

En el segundo capítulo se explica el contexto en el que se realiza el trabajo, las diferencias entre la criptografía simétrica y asimétrica, los requisitos que debe cumplir un sistema seguro, los problemas actuales a la hora de autenticar usuarios en la red, y la Seguridad Basada en la Identidad como solución.

En el tercer capítulo se presenta el análisis del proyecto software llevado a cabo para este TFG.

En el cuarto capítulo, se explica el diseño realizado para el proyecto software, justificando las distintas decisiones tomadas y presentando distintos diagramas de apoyo.

En el quinto capítulo se comenta la implementación del proyecto software, repasando todos los componentes importantes del diseño y explicando cómo se han implementado. Además, se incluye un apartado en el que se explican de forma detallada algunos aspectos programáticos de mayor importancia.

El sexto capítulo plantea las pruebas realizadas para asegurar el funcionamiento del proyecto software, explicando cómo fueron diseñadas y cómo y cuándo se llevaron a término.

Por último, en el séptimo capítulo se presentan las conclusiones, comentando los objetivos principales, repasando lo que se ha aprendido durante el desarrollo del trabajo y haciendo un listado de las posibles mejoras aplicables en un futuro.

Para terminar, existen una serie de anexos con información de interés para el trabajo con temas variados, que van desde la planificación del proyecto, hasta los pasos seguidos para la creación de certificados SSL.

Capítulo 2

Estado del arte

En este capítulo se presentarán los distintos tipos de criptografía, así como los problemas existentes a la hora de diseñar un sistema seguro orientado a asociar diversas identidades digitales a las distintas identidades físicas. En este sentido, se procederá a analizar los pros y los contras de las principales soluciones criptográficas a tal efecto, esto es, los modelos horizontales de confianza en la línea de PGP [12], la arquitectura PKI del estándar X.509 [13], y la arquitectura IBC. En esta exposición el objetivo primordial no es sino presentar IBC como una alternativa tanto a PGP como al estándar X.509.

2.1. Tipos de criptografía

La criptografía es el estudio de las técnicas para conseguir una comunicación segura en un medio no seguro. Mediante el cifrado del mensaje con una clave, se consigue que el mensaje sea ininteligible hasta que es descifrado con otra clave (Ver Figura 2.1). En las secciones siguientes, se hará uso de dos tipos de criptografía diferentes, la criptografía **simétrica** y la criptografía **asimétrica**. A continuación se introducen los principios y fundamentos de esas dos grandes familias de procedimientos criptográficos.

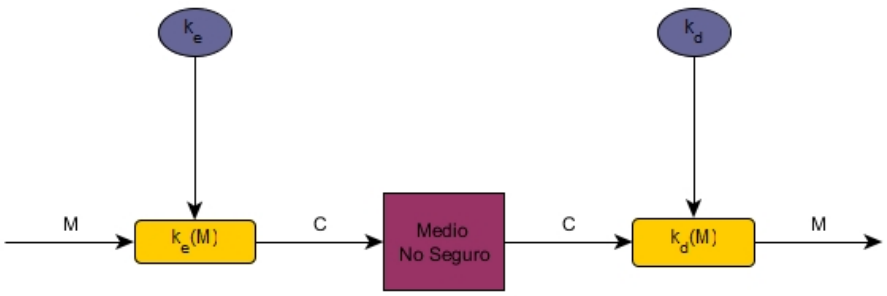


Figura 2.1: Esquema básico de cifrado/descifrado.

2.1.1. Criptografía simétrica

La criptografía simétrica es un tipo de criptografía cuya principal característica es que la **clave** que se utiliza para cifrar es la misma que se emplea para descifrar. Es decir, que en la Figura 2.1: $k_e = k_d$. Lo que significa que cualquiera que posea la clave empleada y conozca el método con el que se cifró, puede descifrar el mensaje. Motivo por el que tanto el emisor como el receptor deben ponerse de acuerdo de antemano en el método y la clave a utilizar durante la comunicación.

Algunas consideraciones para que este tipo de criptografía sea efectiva son las siguientes.

- La clave debe ser impredecible, de nada vale basarse en una clave secreta, conocida únicamente por los usuarios deseados, si alguien puede llegar a averiguar la clave por su cuenta.

- El sistema criptográfico nunca debe basarse en el desconocimiento del método para cifrar, sólo en la clave. Pues la historia ha probado que pasado el tiempo, el método siempre llega a conocerse.
- Ya que cualquiera que tenga la clave puede descifrar el mensaje, tanto el emisor como el receptor deben asegurarse de ser los únicos que conozcan la clave. Por lo que el método para comunicar la clave es tan importante como la clave en sí.

Este tipo de criptografía, que se emplea desde la antigüedad, se sigue empleando hoy en día a diario en una multitud de sistemas diferentes. Un ejemplo de algoritmo actual de criptografía simétrica es AES. En la actualidad, la criptografía simétrica se emplea para cifrar el grueso de los mensajes, debido a que su coste computacional no es demasiado alto.

2.1.2. Criptografía asimétrica

A diferencia de la criptografía simétrica, la criptografía asimétrica no emplea la misma clave para cifrar y para descifrar, o lo que es lo mismo, en la Figura 2.1: $k_e \neq k_d$. En su lugar, presenta dos claves y funciones de cifrado en un único sentido. De este modo, si se cifra con una, sólo se puede descifrar con la otra. Así, cada entidad posee un par de claves, una privada que sólo conocen ellos, y una pública que conoce el resto del mundo. De esta manera, el resto del mundo, en conocimiento de la clave pública, puede escribir mensajes a un usuario para que sólo él/ella pueda leerlo. A su vez, este usuario puede escribir mensajes y cifrarlos con su clave privada que todos pueden leer, ya que todos poseen su clave pública. En la medida que sólo el emisor podría haber escrito tal mensaje, este procedimiento es equivalente a firmar digitalmente el mensaje.

Este tipo de criptografía, nacida en los años 80 [6], supuso un cambio sustancial en el panorama de la seguridad, ya que permitía intercambiar claves de criptografía simétrica de forma segura. Los algoritmos de criptografía asimétrica, como RSA, tienen un alto coste computacional y un límite de tamaño de los mensajes que pueden cifrar, pero son idóneas para cifrar claves compartidas en criptografía simétrica y para firmar Hashes correspondientes a mensajes.

2.2. Problemas al desarrollar un sistema seguro

Para que un sistema pueda ser considerado seguro, debe cumplir una serie de requisitos [14, Capítulo 3]. A continuación, se expondrán los principios básicos de los sistemas de gestión de la seguridad, así como el modo de asegurar que se cumplan usando tecnologías actuales (ver Figura 2.2).

2.2.1. Confidencialidad

La confidencialidad consiste en que sólo los usuarios pertinentes deben poder conocer el contenido del mensaje. Es decir, en un ejemplo en el que *Alice* envía un mensaje a *Bob*, únicamente *Alice* y *Bob* deben conocer qué pone en el mensaje una vez se haya intercambiado.

Un buen método para conseguir confidencialidad es el uso de la criptografía. En otras palabras, transformar el mensaje en algo ininteligible para cualquier persona y que sólo pueda ser devuelto a su estado inteligible conociendo la clave adecuada. Para esto, se puede emplear un algoritmo de criptografía simétrica y una clave lo bastante grande. En efecto, en cifra simétrica si algoritmo de cifrado ha sido debidamente diseñado, un aumento de la longitud de clave reduce la posibilidad de descifrar el mensaje sin conocer la clave, ya que aumenta el coste computacional de un *ataque por fuerza bruta* (i.e., búsqueda exhaustiva de claves de cifrado).

Sin embargo, el problema de este tipo de algoritmos radica en que ambas partes deben ser los únicos en conocer la clave. Una solución sería que ambas personas se viesen, intercambiasen la clave que fuesen a emplear, se asegurasen de que nadie más la supiese, y luego empezar a intercambiarse mensajes cifrados con dicha clave. Este es un método infalible y útil para situaciones de extrema importancia. Sin embargo, para una red de ordenadores en la que entidades que no se conocen entre sí y que viven en lugares completamente distintos del planeta intercambiando constantemente información, no es una solución viable.

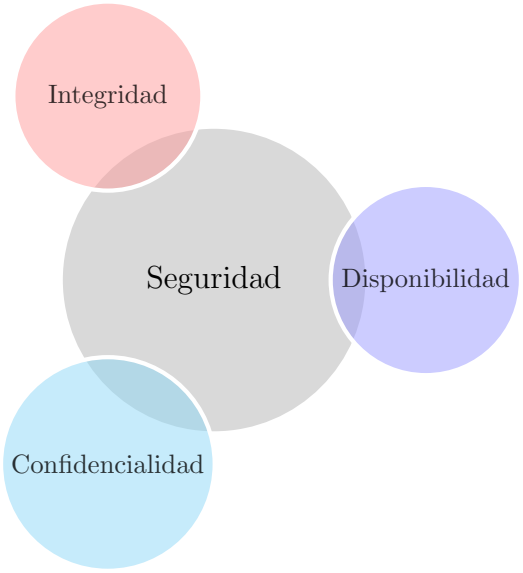


Figura 2.2: Pilares de la Seguridad Informática: Confidencidalidad, Disponibilidad e Integridad.

La solución para el intercambio de claves está en la criptografía asimétrica o de clave pública [15], en la que ambos conocen la clave pública del otro, y ambos tienen una clave privada sólo conocida por ellos. De esta manera, se pueden intercambiar las claves simétricas de forma segura.

2.2.2. Integridad

Sin embargo, no basta con con que los intrusos no puedan leer la información que se intercambia entre varios usuarios, también es necesario saber que nadie ha modificado de algún modo la información que se intercambia. Para esto, se suele acoplar un Hash del mensaje en el propio mensaje, protegido de algún modo, de forma que los receptores puedan generar ellos mismos el Hash, y comprobar que el mensaje no ha sido alterado antes de leerlo.

2.2.3. Disponibilidad

La disponibilidad es un requisito esencial para un sistema seguro, ya que un sistema inaccesible, es completamente seguro, pero también es totalmente inútil. Para ello, el administrador del sistema debe tomar medidas para evitar ataques que puedan impedir la comunicación. Sin embargo, dichas medidas no serán discutidas en este trabajo.

2.2.4. Autenticidad

La autenticidad es el requisito más importante para este proyecto, y consiste en que no sólo debemos estar seguros de que el mensaje pueden leerlo sólo el emisor y el receptor, ni que nadie ha modificado el mensaje, sino que el emisor es quién dice ser. La Autenticidad forma parte de las denominadas Reglas de Oro de los sistemas de seguridad de la información (Autenticación, Autorización y Auditoría), y es muy dependiente de los procedimientos de gestión de identidades digitales. Este problema constituye el núcleo del vigente proyecto. En las secciones siguientes se explicarán tres modos diferentes de abordar este problema.

2.3. Red de confianza

Un modo de asegurar que un usuario es quién dice ser, es mediante los llamados modelos horizontales de confianza, como PGP [12]. Son modelos particularmente simples, al no necesitar de la supervisión de una entidad y estar llevados a cabo únicamente por los usuarios que participan en el sistema. Funciona de tal modo que si un usuario está seguro de que otro usuario es quién dice ser, firma (i.e., cifra con su clave privada) clave pública del nuevo usuario a incorporar en un anillo de confianza. Cuando todos los usuarios de la red hacen lo mismo, se forma una red de confianza en la que si un usuario quiere saber si puede confiar en alguien que no conoce, sólo tiene que buscar a alguien que sí. De este modo, se proporciona un modelo que acaba con el problema de los costes de la PKI, esquema que veremos a continuación.

No obstante, este modelo tiene dos problemas principales. En primer lugar, el esquema depende de los usuarios para estar correctamente implementada (como ocurre con los protocolos Peer to Peer -P2P- en general). Sin embargo, los usuarios pueden ser fácilmente engañados o actuar de manera irresponsable, extendiendo su confianza a otros usuarios que realmente no conocen. En segundo lugar, aquellos usuarios que estén más aislados en la red de confianza son considerados menos creíbles, por lo que no hay igualdad en el sistema.

2.4. PKI

La Infraestructura de Clave Pública, cuyo estándar más relevante es el X.509 [13], es el modelo más utilizado actualmente para asegurar la autenticidad, y provee una arquitectura con dos pilares principales: la **autoridad de registro** y la **autoridad de certificación**. Los usuarios registran sus claves públicas en la **autoridad de registro**, que verifica que el usuario es quién dice ser. Una vez se ha confirmado la autenticidad de las credenciales del usuario, la **autoridad de certificación** emite un **certificado** para su clave pública [4].

De este modo, en una comunicación entre iguales, existe una entidad confiable que actúa como tercero y que asegura las identidades de ambas partes. Este mecanismo de validación esta basado en la firma digital, de forma que la confianza otorgada por una autoridad certificadora no es sino la firma digital, con la clave privada de la autoridad, del Hash de un documento (el certificado) que contiene la clave pública de un usuario e información sobre el mismo. Por lo que, si *Bob* recibe un mensaje firmado por *Alice* con su clave privada, *Bob* mirará que el certificado asociado a la clave pública con la que va a verificar que el mensaje proviene de *Alice*, es verdadero y confiable. Es decir, que el certificado diga que efectivamente *Alice* es quien dice ser, y que la entidad que ha emitido el certificado sea confiable.

La Infraestructura de Clave Pública se extiende de forma jerárquica de acuerdo a un modelo vertical de confianza. Así, existen muchas entidades repartidas que tienen autoridad para registrar usuarios y emitir certificados habiendo obtenido dicha autoridad de otra entidad de mayor confianza. De esta manera, la infraestructura es fácilmente expansible o escalable.

Sin embargo, pese a ser el modelo más empleado para proporcionar autenticidad, posee una serie de desventajas, la más evidente siendo la falta de flexibilidad. Para que un usuario sea confiable, debe registrar previamente su clave pública, y para que el sistema funcione, la entidad debe seguir un estricto protocolo para asegurarse de que el usuario es quién dice ser. Por lo que los procesos de registro y revocación de claves no son automáticos, sino que requieren de supervisión por parte de la entidad, lo cual lleva tiempo (sobre todo en el caso de la revocación de certificados digitales). La Criptografía Basada en la Identidad (IBC) propone una arquitectura que da solución a algunos de estos problemas.

2.5. Seguridad basada en la identidad

En esta sección se explicará el modelo alternativo que propone la Seguridad Basada en la Identidad, comenzando por reseñar cómo funciona, los principales algoritmos necesarios para implementarla, sus ventajas y sus inconvenientes.

2.5.1. Fundamentos

La seguridad basada en la identidad nace del deseo de simplificar la gestión de certificados [16]. La arquitectura de la IBC gira alrededor de una entidad altamente confiable llamada Private Key Generator (PKG), que genera un par de claves pública - privada llamadas las claves maestras. Los usuarios se registran en la PKG y generan sus claves bajo demanda, en lugar de registrar una única clave al registrarse la primera vez.

Consecuentemente la comunicación entre *Alice* y *Bob* cambia: cuando *Alice* quiere mandar un mensaje a *Bob*, ya no es necesario que *Alice* obtenga el certificado de la clave pública de *Bob*, sino que sencillamente puede utilizar su identidad y la clave pública maestra para cifrar un mensaje que sólo *Bob* pueda leer. Además, puede hacer esto incluso antes de que *Bob* haya generado sus claves. La identidad es una cadena normalmente fácil de recordar como puede ser el e-mail o el número de teléfono del usuario. Cuando *Bob* quiera leer el mensaje, se autenticará a la PKG y pedirá que se le genere una clave privada. Con esta clave, *Bob* podrá leer el mensaje de *Alice* (ver Figura 2.3). La seguridad del sistema se basa por lo tanto en que *Alice* sabe que sólo *Bob* puede acceder a la PKG como Bob y generar su clave privada para leer el mensaje.

El gran cambio respecto a la PKI es que las claves se generan bajo demanda y, por tanto, el despliegue del modelo de confianza es más flexible. Otro cambio importante es que la PKG tiene acceso a las claves privadas de los usuarios, por lo que podría hacerse pasar por ellos sin ninguna dificultad. Por tanto, la PKG debe ser una entidad altamente confiable, de otro modo este modelo no tendría sentido.

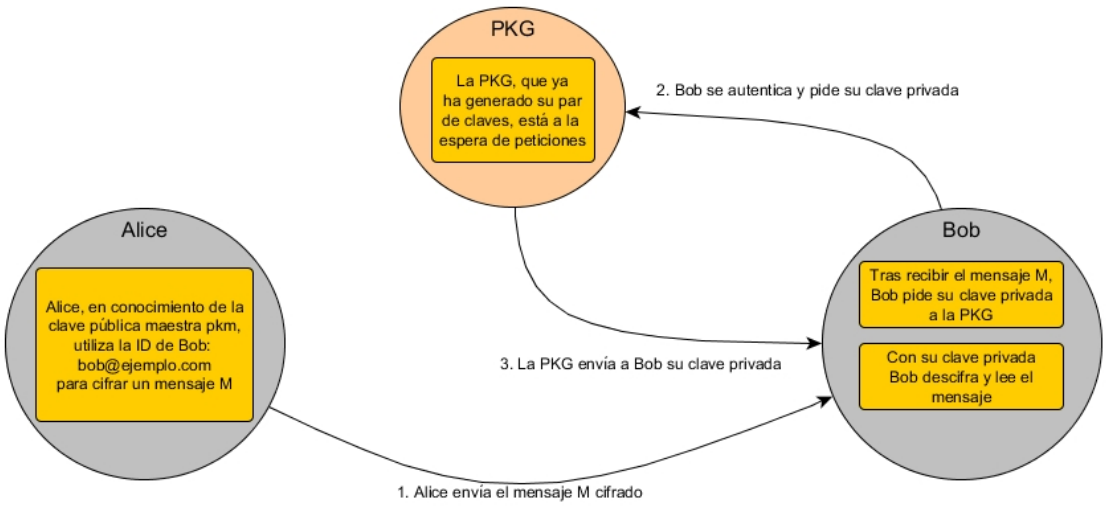


Figura 2.3: Ejemplo de comunicación básico con IBE.

2.5.2. Algoritmos

Los principales algoritmos del modelo IBC son los siguientes [16].

Setup Genera los parámetros del sistema, que se harán públicos, (los llamaremos clave pública maestra), y la clave privada maestra, a la que sólo tendrá acceso la PKG.

Extract Genera una clave privada a partir de las claves maestras y el identificador del usuario, como por ejemplo, su e-mail.

Encrypt Con la clave pública maestra, el identificador del destinatario y el mensaje, devuelve el mensaje cifrado.

Decrypt Con la clave pública maestra, el mensaje cifrado y una clave privada adecuada, devuelve el mensaje original.

2.5.3. Ventajas

Debido a su arquitectura, la Seguridad Basada en la Identidad ofrece muchas ventajas, algunas de las más importantes se exponen a continuación.

1. Al eliminar la entidad certificadora, se reducen los costes de mantenerla [6].
2. En un proceso de comunicación completo, sólo se debe acceder una única vez a la PKG, esto es, el receptor a la hora de obtener su clave privada, lo cual reduce la complejidad y acelera el proceso.
3. Las claves privadas se generan bajo demanda. Al no registrar una única clave pública, y al tener la PKG acceso a las claves privadas, la revocación de claves y la recuperación de las mismas son muy fáciles de llevar a cabo [16]. Tanto la revocación como la recuperación de claves son procesos muy complicados en el esquema PKI.
4. Permiten tratar fácilmente con grupos de usuarios, permitiendo, entre otros, firmas grupales [4].
5. Al tener una alta flexibilidad para generar claves, se pueden generar con fecha de caducidad diaria, ideal para crear claves efímeras, o para delegar la capacidad de descifrado a otras entidades durante un tiempo limitado [9], [16].

2.5.4. Inconvenientes

Sin embargo, pese a todas sus ventajas, IBC tiene sus inconvenientes. El principal inconveniente está provocado por su entidad principal, la PKG, que tiene demasiada responsabilidad y presenta un punto único de fallo (Single Point of Failure -SPOF-) en la arquitectura. Además, la PKG necesita un altísimo grado de confianza y requiere unas medidas de seguridad excepcionales, ya que al almacenar las claves privadas de los usuarios, tiene toda la información que necesita para hacerse pasar por cualquiera de ellos, por lo que si llegase a ver una brecha de seguridad en la PKG, se comprometería toda la información cifrada con dichas claves. Es es el problema conocido como *key scrow*. Existen variantes de la IBC que eliminan el problema del *key scrow* dividiendo del proceso de generación de claves entre la PKG y el usuario [17]. Sin embargo, no se entrará en detalles en cuanto a estos modelos se refiere.

Además, los usuarios necesitan autenticarse contra la PKG de algún modo, e.g., mediante autenticación basada en un par nombre de usuario y contraseña. Por lo que cada usuario debe velar por su contraseña ya que si un atacante es capaz de averiguarla, puede hacerse pasar por dicho usuario sin que la PKG, ni el resto de usuarios puedan enterarse.

Capítulo 3

Análisis

A lo largo de este capítulo se procederá a realizar un análisis del sistema propuesto, detallando sus objetivos principales y su alcance. Además, se incluirá la lista completa de requisitos funcionales y no funcionales, así como el diagrama de casos de uso. La planificación temporal del proyecto se puede consultar en el Anexo A.

3.1. Definición del proyecto

El primer estadio de la concreción del proyecto viene dado para la clarificación de la aplicación, así como especificar sus objetivos principales, las funcionalidades desarrolladas y su alcance.

3.1.1. Objetivos y funcionalidad

El objetivo principal de este proyecto, tal y como dice su propio título es el **desarrollo de un esquema de autenticación basado en la identidad para Clientes Android**. Para cumplir este objetivo se necesitan tanto un Cliente Android, como la infraestructura necesaria para asegurar que el esquema de autenticación esté basado en la identidad.

El sistema propuesto para cumplir ambos objetivos es el de un Cliente Android de mensajería que permita la comunicación escrita entre usuarios de manera instantánea. Los mensajes están cifrados de extremo a extremo utilizando IBE de manera completamente transparente para el usuario. Además, el usuario podrá crear grupos o eliminar grupos existentes, añadir a otros usuarios a grupos existentes y escribir y recibir mensajes de ellos. Para ello, la aplicación se comunicará con un servidor externo (PKG) que gestionará las claves y asegurará que el esquema esté basado en la identidad.

Se necesitan por lo tanto, dos aplicaciones independientes que responden a dos tipos de usuario completamente distintos, por un lado la aplicación de mensajería en Android, que será utilizada por usuarios registrados para comunicarse entre ellos de forma segura. Y por otro lado, el Servidor de Aplicaciones, que responderá sólo al administrador del sistema y que se encargará de gestionar las claves y de almacenar toda la información necesaria para que pueda llevarse a cabo la autenticación y el intercambio de mensajes.

3.1.2. Alcance

Servidor El administrador de la aplicación ya está registrado y debe hacer login para iniciar/-parar el servidor. No se puede registrar a ningún otro administrador. El servidor permite generar nuevos pares de claves maestras, pero no le está permitido al administrador, ya que se dejarían de poder leer todos los mensajes generados hasta entonces.

Aunque el servidor deba implementar la mayoría de las funcionalidades necesarias para que el administrador pueda realizar algunas acciones, como generar otro par de claves maestras cuando quiera, no será un requisito ya que esto ocasionaría la necesidad de realizar cambios a lo largo de todo el servidor, los cuales están fuera del alcance de este proyecto. Es por ello que el administrador sólo puede iniciar y parar el servidor.

Cliente Android Cualquier usuario registrado puede crear una nueva conversación con alguien, y escribir y recibir mensajes en ella. Puede también crear grupos, añadir participantes al grupo (si es el creador), ver sus participantes, borrar el grupo (si es el administrador) y ver las conversaciones activas ordenadas, estando primero aquellas con mensajes no leídos.

El propósito de la aplicación Android es proveer un entorno realista que pueda cumplir las funcionalidades básicas de un servicio de mensajería y sobre todo, permitir probar que el sistema total funciona como es debido, especialmente la parte de IBE. Es por esto que se han dejado fuera algunas funcionalidades, como quitar usuarios de los grupos, poder añadir varios participantes a la vez o poder añadirlos al crear el grupo.

Todas estas funcionalidades añaden valor y usabilidad a la aplicación, sin embargo, al no ser una parte esencial de los objetivos, se han dejado fuera del alcance del proyecto (Ver sección 7).

3.2. Catálogo de requisitos

En esta sección se detalla el listado de requisitos funcionales y no funcionales del proyecto.

3.2.1. Requisitos funcionales

A continuación se listan, en primer lugar, los requisitos funcionales del servidor, y en segundo lugar los del Cliente Android.

Servidor

RF1 Solo el administrador podrá iniciar el servidor.

RF2 Se llevará registro de todos los usuarios registrados y su información necesaria.

RF3 Se podrán generar las claves maestras en caso de que no hayan sido generadas.

RF4 Un usuario podrá registrarse proporcionando un nombre de usuario y contraseña válidos.

RF4.1 El nombre de usuario es único y no contiene caracteres extraños.

RF4.2 La contraseña tendrá una longitud superior a 8 y deberá contener mayúsculas, minúsculas, números y símbolos permitidos.

RF5 Un usuario podrá autenticarse con su nombre de usuario y contraseña.

RF6 Cualquier usuario autenticado podrá obtener su clave privada.

RF7 Se llevará registro del historial de claves generadas por un usuario.

RF8 Cualquier usuario autenticado podrá obtener su historial de claves.

RF9 Un usuario autenticado podrá obtener un listado con los mensajes que no ha leído.

RF10 Un usuario autenticado podrá informar al servidor de que ha leído un determinado mensaje y el servidor actualizará la información pertinente.

RF11 Una vez iniciado, el servidor podrá ser detenido en cualquier momento.

Cliente Android

RF1 El usuario podrá registrarse en la aplicación.

RF2 El usuario podrá autenticarse una vez se haya registrado en la aplicación.

RF3 Una vez autenticado, el usuario podrá visualizar su lista de conversaciones activas.

RF3.1 Las conversaciones con mensajes nuevos se mostrarán antes que el resto de conversaciones.

RF4 El usuario podrá acceder a cualquiera de sus conversaciones activas y leerlas en su totalidad.

RF5 El usuario podrá crear una nueva conversación con cualquier otro usuario.

RF6 El usuario podrá crear un grupo .

RF7 El usuario podrá añadir participantes al grupo.

RF7.1 El usuario deberá ser el creador del grupo para poder añadir nuevos miembros.

RF8 El usuario podrá eliminar un grupo.

RF8.1 El usuario deberá ser el creador del grupo para poder eliminarlo.

RF9 Una vez en un chat, ya sea grupal o individual, el usuario podrá enviar mensajes.

RF10 Una vez en un chat, el usuario recibirá los mensajes que escriban los otros usuarios en tiempo real.

RF11 Incluso si la aplicación no está en primer plano, el usuario recibirá notificaciones de los nuevos mensajes que le lleguen en tiempo real.

RF12 El uso de criptografía para asegurar la seguridad de los mensajes será totalmente transparente al usuario.

RF13 El usuario sólo deberá iniciar sesión una única vez, cada vez que inicie la aplicación.

RF14 El usuario podrá detener la aplicación en cualquier momento.

3.2.2. Requisitos no funcionales

Seguridad

RNF1 Todas las contraseñas almacenadas estarán debidamente cifradas.

RNF2 Toda la información enviada a través de la web estará debidamente cifrada.

RNF3 Sólo usuarios registrados podrán acceder a la aplicación.

Eficiencia

RNF4 La aplicación Cliente Android deberá ser lo más ligera posible, para economizar el uso de la batería al máximo.

RNF5 La interfaz del servidor no es importante debido a la escasa cantidad de acciones que puede realizar el administrador. Deberá mantenerse al mínimo para maximizar la eficiencia.

Usabilidad

RNF6 La interfaz de usuario de la aplicación Android será lo más intuitiva posible, para ello, utilizará una interfaz parecida a las aplicaciones de mensajería más comunes.

Sistema

RNF7 El Cliente Android deberá funcionar en dispositivos con una API de versión 15 o superior.

3.3. Casos de uso

En esta sección se presentarán los casos de uso del proyecto, haciendo siempre una distinción entre el Servidor y el Cliente Android. Se comenzará con una breve explicación y el diagrama de casos de uso. Para ver dos ejemplos de casos de uso concretos, se recomienda visitar el Anexo B.

A continuación se puede observar el diagrama de casos de uso, donde se muestran representadas las posibles acciones mencionadas en el anterior catálogo de requisitos, para cada uno de los actores que forman parte del proyecto: el administrador y el usuario (Ver Figura 3.1).

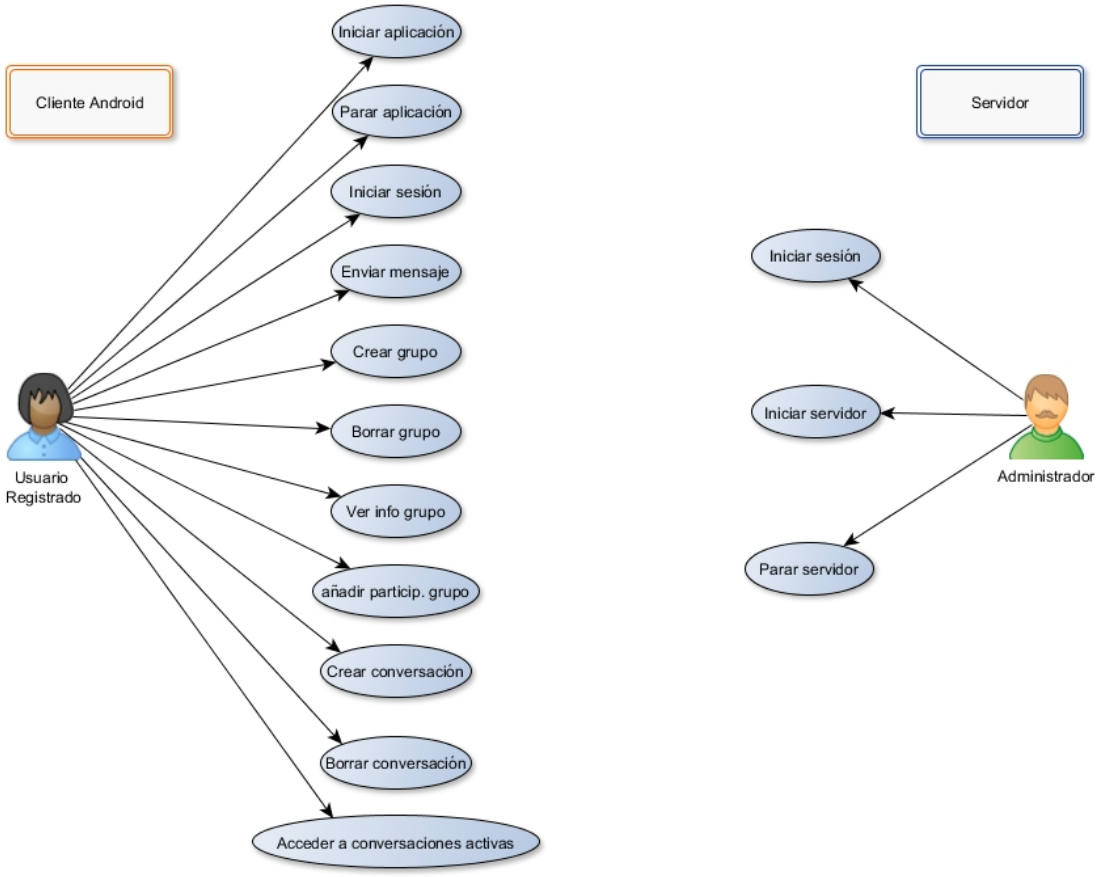


Figura 3.1: Diagrama de casos de uso.

Por último, mostramos dos casos de uso concretos: el de inicio del Servidor (Figura B.1) y el de envío de mensaje en el Cliente Android (Figura B.2) que se pueden ver en el Anexo B.

Capítulo 4

Diseño

El foco de este capítulo será mostrar el diseño de la aplicación, haciendo hincapié en explicar el por qué de las decisiones tomadas. Se detallará en profundidad la arquitectura del sistema, los requisitos técnicos para poder poner en marcha el proyecto y la interacción entre los distintos componentes que forman la arquitectura. También se incluirá un apartado explicando ciertas limitaciones técnicas que influyeron en el diseño de la aplicación.

4.1. Arquitectura del sistema

El propósito de esta sección es extraer del análisis los principales componentes que formarán la arquitectura de la aplicación, discutir los diferentes modos de conectarlos, mostrar ejemplos y explicar la elección final que se tomó para el proyecto y por qué.

4.1.1. Componentes principales

Del análisis en profundidad de los requisitos se extrajeron una serie de componentes indispensables para el diseño de la aplicación:

- Se requiere una aplicación de mensajería basada en Android, por lo que uno de los componentes obligatorios es el **Cliente Android**.
- La autenticación debe seguir un modelo basado en la identidad, y siguiendo las pautas de la Encriptación Basada en la Identidad, necesitamos una tercera parte altamente confiable que se encargue de generar las claves. Otro componente indispensable es por tanto la Private Key Generator (PKG).
- Se va a establecer un servicio de mensajería entre usuarios, para llevar esto a cabo, se elige un modelo con un Servidor de Aplicaciones, que guarde registro de todos los usuarios, de todos los mensajes, y de los grupos de usuarios que se formen. El tercer elemento imprescindible es por lo tanto el **Servidor de Aplicaciones**.

4.1.2. Conexión entre componentes

Una vez identificados los principales componentes de nuestra arquitectura, queda ver cómo se conectan entre ellos. Como se puede observar de los componentes ya identificados, son componentes con una función individual muy clara, lo cuál va a permitir establecer un **diseño estructurado** sin demasiado esfuerzo. Para ello, se elige una **arquitectura multicapa** con una estructura altamente **modular** donde cada uno de los módulos sea lo más independiente posible. La estructura modular tiene la ventaja de apenas mostrar **acoplamiento** entre sus módulos, sin embargo, la **cohesión** debe ser lo más alta posible. Teniendo todo esto en cuenta, la solución viene dada por servidores REST, de este modo, cada servidor puede ser un módulo **independiente** con libertad de implementación y tamaño y puede comunicarse con el resto únicamente haciendo las peticiones HTTP adecuadas. Además, ofrece la posibilidad de tener cada módulo en un equipo distinto, convirtiéndolo sin esfuerzo en una aplicación distribuida.

4.1.3. Primer diseño de arquitectura

De acuerdo a las consideraciones anteriores, la arquitectura propuesta inicialmente es la que se ve en la Figura C.1 en el anexo C. Como se puede observar en dicha Figura, una serie de Clientes Android se conectan a través de internet al Servidor de Aplicaciones, que es el que se encarga de almacenar toda la información en una base datos. Sería este mismo servidor el que se comunicaría con la PKG cuando fuese necesario para dar respuesta a las peticiones de los usuarios. Es importante hacer notar que la PKG se comunica únicamente con el Servidor de Aplicaciones, no aceptará peticiones de ninguna otra entidad.

Siguiendo esta arquitectura, un ejemplo significativo del uso de la aplicación sería el siguiente (se proveerá un esquema más concreto y detallado de las técnicas utilizadas para cada parte en el capítulo 5:

1. El Cliente se registra/inicia sesión en el Servidor de Aplicaciones mediante una petición HTTP.
2. El Servidor de Aplicaciones guarda/consulta la información en la base de datos y da una respuesta.
3. Si es la primera vez, la aplicación solicita al Servidor de Aplicaciones que genere un par de claves de usuario.
4. El Servidor de Aplicaciones actúa de intermediario entre la PKG y el Cliente mandando la petición a la PKG y devolviendo la respuesta al Cliente.
5. El Cliente cifra el mensaje con criptografía simétrica, cifra la clave con IBE con la identidad del destinatario, hace un Hash del mensaje, lo firma y lo envía por SSL al Servidor de Aplicaciones.
6. El Servidor de Aplicaciones, recibe el mensaje y lo guarda en la base de datos.
7. El Cliente de destino, cada cierto tiempo, pregunta al Servidor de Aplicaciones si tiene mensajes nuevos.
8. Ante esta petición, el servidor le mandaría el mensaje cifrado.
9. El Cliente, una vez recibido el mensaje y sabiendo de quién proviene, pediría al Servidor de Aplicaciones su propia clave privada.
10. De nuevo, el Servidor de Aplicaciones actuaría de intermediario entre la PKG y el Cliente.
11. Por último, el Cliente comprobaría que está firmado por quien dice ser, haría el Hash del mensaje total y comprobaría que no ha sido alterado, con su clave privada descifraría la clave usada para cifrar el mensaje y descifraría el mensaje total.

4.1.4. Consideraciones de diseño

Una vez propuesta la arquitectura, se decidieron las tecnologías más adecuadas para llevar a cabo el proyecto.

- El punto más importante era decidir la biblioteca que se encargase de la Criptografía Basada en la Identidad. Se consideraron dos opciones: jPBC [18] y Charm [7] para los lenguajes de programación Java y Python respectivamente. Se terminó escogiendo Charm debido a una mejor documentación de sus funciones y que incluía una forma muy sencilla de serializar las claves y objetos criptográficos [19], algo esencial al necesitar transferirlos a través de la web.
- De igual importancia era decidir si utilizar Java y Spring [20] o si utilizar Python y Flask [10]. Se terminó eligiendo Python y Flask por su mayor facilidad de aprendizaje y por proveer una implementación más directa, con menos ficheros de configuracion y menos líneas de código que Java y Spring. Por supuesto, el hecho de preferir Charm a jPBC también influyó en la decisión.

Además de elegir las tecnologías, y debido a la importancia que se da a la seguridad en este proyecto además de responder directamente a algunos requisitos propuestos en el análisis, se tomaron otras consideraciones respecto de la seguridad de la aplicación.

- Las contraseñas de los usuarios se almacenarán en la base de datos después de pasarla por una función Hash poniendo esfuerzo extra para que no sean débiles contra ataques de diccionario, y forzando a los usuarios a escoger contraseñas seguras.
- Los mensajes de los usuarios se almacenarán cifrados.
- Todas las comunicaciones entre componentes estarán debidamente cifradas con SSL.
- Las claves IBE de los usuarios se almacenarán en el servidor, pero estarán cifradas a partir de la contraseña de los propios usuarios, contraseña que el servidor no almacenará, de modo que sólo cuando el usuario introduzca la contraseña, pueda el servidor acceder a su contenido.

4.1.5. Limitaciones

La arquitectura anterior es perfectamente válida y durante las primeras fases del desarrollo fue considerada como la arquitectura final. Sin embargo, a día de hoy, existen ciertas limitaciones con las tecnologías elegidas para desarrollar el proyecto que obligaron a reconsiderar el diseño una vez se había comenzado a implementar. Como ya se ha comentado, la idea original era cifrado de extremo a extremo, lo que significa que serían los dispositivos ejecutando el Cliente Android los que cifrarían los mensajes, y estos mensajes serían almacenados en el servidor todavía cifrados. De tal manera que sólo los Clientes emisor y receptor pudiesen ver el mensaje descifrado.

Sin embargo, a día de hoy no existen métodos maduros establecidos para ejecutar Python en Android, y sobre todo, la biblioteca Charm está todavía en fase de implementación, y conseguir que funcione en Android es relativamente complicado. La aplicación de Cliente Android está pensada para ser totalmente transparente para los usuarios, sin embargo para poder cifrar de extremo a extremo se requeriría instalar aparte la biblioteca Charm en Android, lo cual requiere de unos conocimientos avanzados. Por este motivo, se decidió que el dispositivo Android no utilizaría Charm.

Aun así, Android funciona sobre java, luego todavía existía la posibilidad de que el servidor proporcionase las claves utilizando Charm, y el móvil las utilizase empleando la biblioteca jPBC. Sin embargo, al no existir una forma de compatibilizar ambas bibliotecas se decidió finalmente que el Cliente Android no utilizaría IBE.

4.1.6. Diseño final de arquitectura

Al presentarse las limitaciones que se muestran en la sección anterior, se hace necesaria la creación de una nueva arquitectura, ya que si el dispositivo móvil no puede utilizar IBE, debe existir un nuevo componente que lo haga por él. Surge por lo tanto un nuevo componente resultante de dividir el Servidor de Aplicaciones en dos nuevos componentes: el servidor encargado de gestionar la base de datos y el servidor que gestiona todas las peticiones del Cliente y cifra y descifra los mensajes por él.

Se puede ver el diagrama de la nueva arquitectura en la Figura 4.1, en el que se puede observar que es el nuevo servidor el que trata con los Clientes y con el servidor de la base de datos. Veremos más en profundidad las funcionalidades específicas de cada módulo y la conexión entre módulos en la sección 4.2.

Con esta nueva arquitectura, el esquema de comunicación de antes quedaría renovado, y es que ahora el Cliente no haría nada relacionado con la cifrado, enviaría sus mensajes en claro a través de SSL al Servidor, que los cifraría y enviaría al Servidor de Base de Datos para su almacenamiento. Al recibir, el Servidor descifraría los mensajes y los enviaría al Cliente destinatario tras recibir una petición del Cliente.

Con este nuevo diseño surge la duda de si la aplicación sigue dando respuesta a los objetivos iniciales. Al no haber ya cifrado de extremo a extremo estamos ante una situación un poco irregular, y es que, aunque sean Servidores separados, es el mismo Servidor el que está cifrando y descifrando la información, entonces, ¿para qué la ciframos?

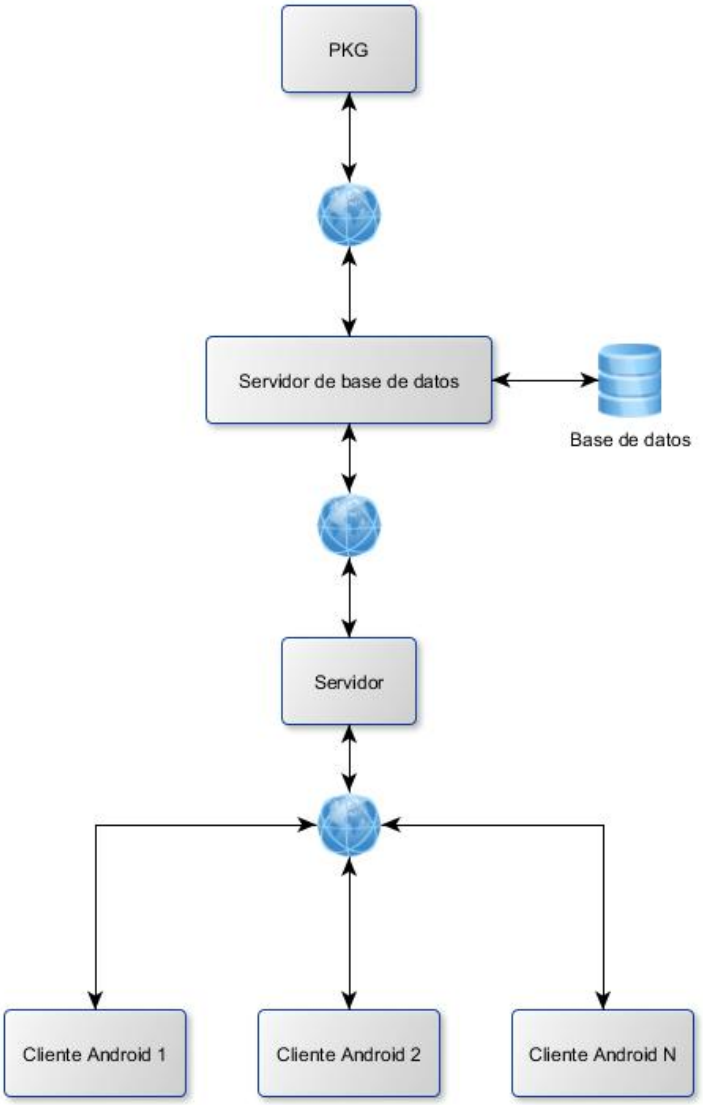


Figura 4.1: Diagrama de la arquitectura final del proyecto, en el que se ven los cuatro componentes principales: la PKG, el servidor de gestión de base de datos, el servidor que recibe todas las peticiones de los Clientes y cifra/descifra los mensajes y por último los Clientes.

Uno de los objetivos principales de este proyecto es el de realizar un sistema de autenticación basado en la identidad, de esta forma, aunque el cifrado no vaya de extremo a extremo (protegida por IBE, ya que toda la comunicación está protegida por SSL, todas las funcionalidades de IBE están desarrolladas e implementadas, lo cuál cumple el objetivo principal y añade las siguientes ventajas:

- Se elimina todo el cifrado en el Cliente, lo cual es bueno, ya que una de los principales puntos débiles de los dispositivos móviles es su escasa batería. Al eliminar el cifrado, un proceso computacionalmente caro, también se reduce el uso de batería, convirtiendo el dispositivo móvil en poco más que una interfaz de visualización. Lo cual está también de acuerdo con el requisito **RNF4** visto en la sección de análisis 3.2.2.

- El componente al que llamamos Servidor y que se comunica directamente con el Servidor de Base de Datos es un componente intermedio porque la aplicación está diseñada para que el Cliente esté en un dispositivo Android. Pero ahora que cifra y descifra, el componente Servidor, podría ser perfectamente un Cliente de mensajería en un ordenador con Charm instalado, que podría comunicarse con otros Clientes sin ninguna funcionalidad adicional. Sólo necesitaría una interfaz de usuario. Ver sección 7.

- Escalabilidad. Al ser una arquitectura totalmente modular, es fácilmente escalable y distribuible. El componente con más carga de trabajo de toda la aplicación, es el nuevo Servidor, que recibe todas las peticiones del Cliente y además se encarga de cifrar y descifrar. Sin embargo, dicho componente podría ser fácilmente replicado, y acceder a sus servicios mediante un balanceador de carga. Ver anexo D para más detalles.

4.2. Interacción entre componentes

A lo largo de esta sección se explicará la interacción entre los componentes de la arquitectura propuesta, comenzando por lo general y terminando con un ejemplo específico de funcionamiento del sistema que ilustra especialmente bien la comunicación entre los distintos componentes que lo componen.

4.2.1. Idea general

La idea general de la interacción entre componentes es muy sencilla, ya que cada componente tiene una funcionalidad clara y definida y la forma de comunicarse entre todos los componentes es la misma: peticiones HTTP. Por lo tanto el mecanismo de funcionamiento es siempre igual: el Cliente necesita algo, luego realiza una petición que es atendida por el Servidor, que dependiendo de si puede resolverla por si mismo o si por el contrario requiere de los servicios de otro de los servidores, trata la petición y contesta o relega la información hacia los otros servidores, actuando de intermediario. Por lo tanto, y como es obvio, cada servidor se encargará de las tareas que sean una petición directa a su funcionalidad.

- PKG, genera las claves maestras IBE la primera vez que se inicia, y genera claves para los usuarios a partir de su nombre de usuario bajo demanda.

- Servidor de Base de Datos, maneja toda las funcionalidades que tienen que ver con la conexión con la base de datos. Por lo tanto, todas las petición de registro, inicio de sesión, almacenar un nuevo mensaje, comprobar si se tienen nuevos mensajes, etc. son llevadas a cabo por este Servidor. Como es el único componente que puede comunicarse con la PKG, también se encarga de transferirle las peticiones pertinentes y devolver su respuesta.

- Servidor, su función principal es el de cifrar los mensajes que llegan del Cliente y descifrar los mensajes antes de enviarlos recibidos antes de enviarlos a los Clientes correspondientes. Pero al recibir todas las peticiones, también es el encargado de transferirlas al Servidor de Base de Datos cuando la petición está fuera de su jurisdicción.

- Cliente, es el generador de todas las peticiones las cuales hace según necesite. A sus ojos, sólo existe un servidor, que es con el que se comunica y el que responde a todas sus peticiones, aunque como ya hemos visto, esto no es así.

4.2.2. Ejemplo detallado de envío y recepción de mensajes

Después de ver la idea general, se muestra un ejemplo concreto para ilustrar de forma más específica el funcionamiento global de la aplicación. A continuación se muestra una lista con las diferentes etapas por las que pasa la aplicación al cumplir una de sus funcionalidades básicas, el envío y la recepción de mensajes.

1. El Cliente envía una petición de registro/inicio de sesión al Servidor.
2. El Servidor actúa de intermediario entre el Servidor de Base de Datos y el Cliente y comunica la respuesta del Servidor de Base de Datos al Cliente.
3. Si es la primera vez, la aplicación solicita al Servidor que genere un par de claves de usuario.
4. El Servidor reenvía la petición al Servidor de Base de datos que actúa de intermediario entre la PKG y el Servidor mandando la petición a la PKG y devolviendo la respuesta al Servidor, que comunica la respuesta al Cliente.
5. El Cliente quiere enviar un mensaje a otro Cliente, así que envía el mensaje en texto plano al Servidor.
6. El servidor solicita las claves del Cliente al Servidor de Base de Datos, que como ya han sido generadas, las tiene almacenadas. Cifra el mensaje con criptografía simétrica, cifra la clave con IBE utilizando el nombre de usuario del destinatario, hace un Hash del mensaje, lo firma con la privada del emisor y lo envía por SSL al Servidor de Base de Datos.
7. El Servidor de Base de Datos, recibe el mensaje y lo guarda en la base de datos.
8. El Cliente de destino, cada cierto tiempo, pregunta al Servidor si tiene mensajes nuevos, éste, le pregunta al Servidor de base de Datos que ante esta petición, le mandaría el mensaje cifrado.
9. El servidor, viendo que hay un mensaje nuevo, pediría de nuevo las claves IBE al Servidor de Base de Datos, y con ellas descifraría la clave simétrica, con la que descifraría el mensaje y se lo enviaría descifrado al Cliente.
10. El Cliente, una vez recibido el mensaje, simplemente lo mostraría.

4.3. Diseño del Cliente Android

Hasta ahora, el diseño se había centrado en la aplicación en general, donde los componentes más específicos eran los de la parte del servidor, y el Cliente Android era tratado como una caja negra, que cumplía su función sin saber qué había dentro. Habiendo terminado la parte del Servidor, es el momento de centrarse en el Cliente, cuyo diseño es particularmente importante, ya que será el componente que entre en contacto directo con el usuario. A continuación se explicará el diseño de la aplicación Android desde el aspecto lógico y desde el aspecto gráfico.

4.3.1. Diseño lógico

El diseño de la aplicación para móvil se centra en responder directamente las necesidades del Cliente vistas en la sección de análisis (ver sección 3). En la siguiente Figura (ver Figura 4.2) se puede observar el diagrama con las actividades de la aplicación, su flujo de interacción y sus principales funcionalidades.

Como se puede ver en el diagrama, todas las actividades parten de la necesidad de iniciar sesión. Esto es algo normal, ya que todas las funcionalidades del servidor requieren que el usuario

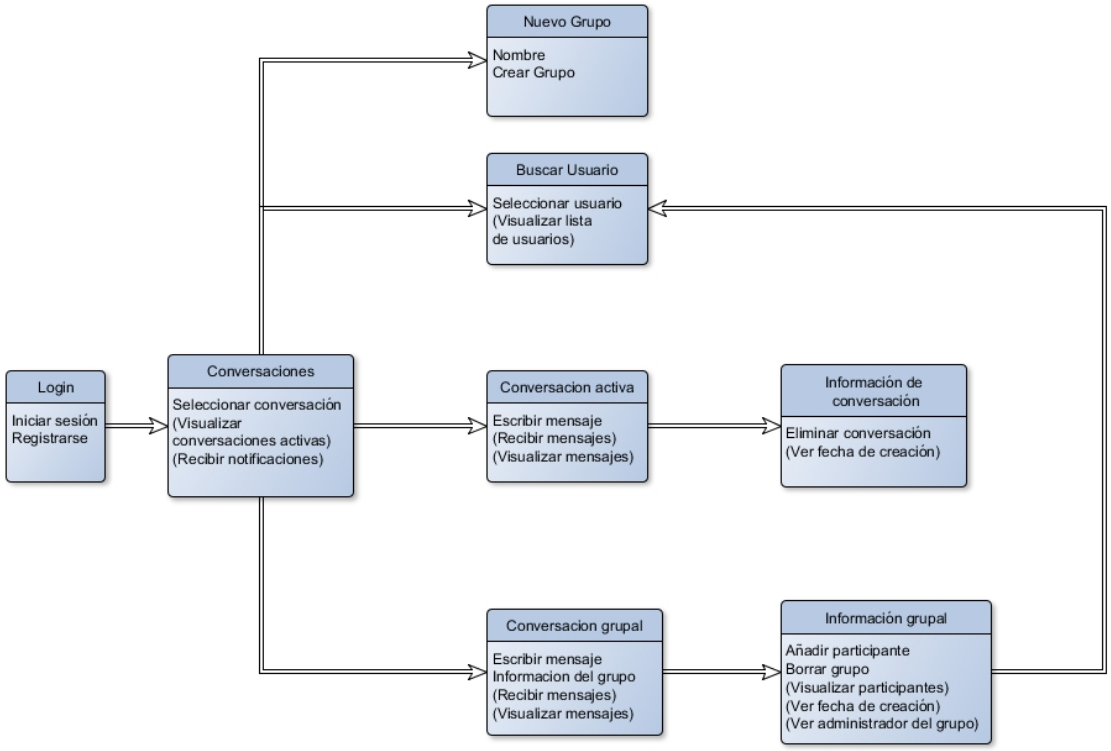


Figura 4.2: Diagrama que muestra las distintas "pantallas.º actividades del Cliente, sus funcionalidades básicas y su flujo de interacción.

provea su nombre usuario y contraseña. Una vez introducidas, se almacenará en el teléfono para ser usadas cuando sea necesario, sin que el usuario deba introducirlas de nuevo, convirtiendo el uso de la aplicación en una experiencia más confortable.

Una vez el usuario ha iniciado sesión, procede a la que será la actividad principal de la aplicación móvil, la que es denominada actividad de **conversaciones**. Esta actividad, de manera muy similar a otras aplicaciones de mensajería móvil como Telegram [21] o Whatsapp [22], muestra todas las conversaciones activas del usuario ordenándolas de más a menos reciente, y mostrando algún tipo de señalización cuando haya mensajes sin leer. Desde esta actividad se podrán realizar el grueso de las acciones que puede realizar el usuario:

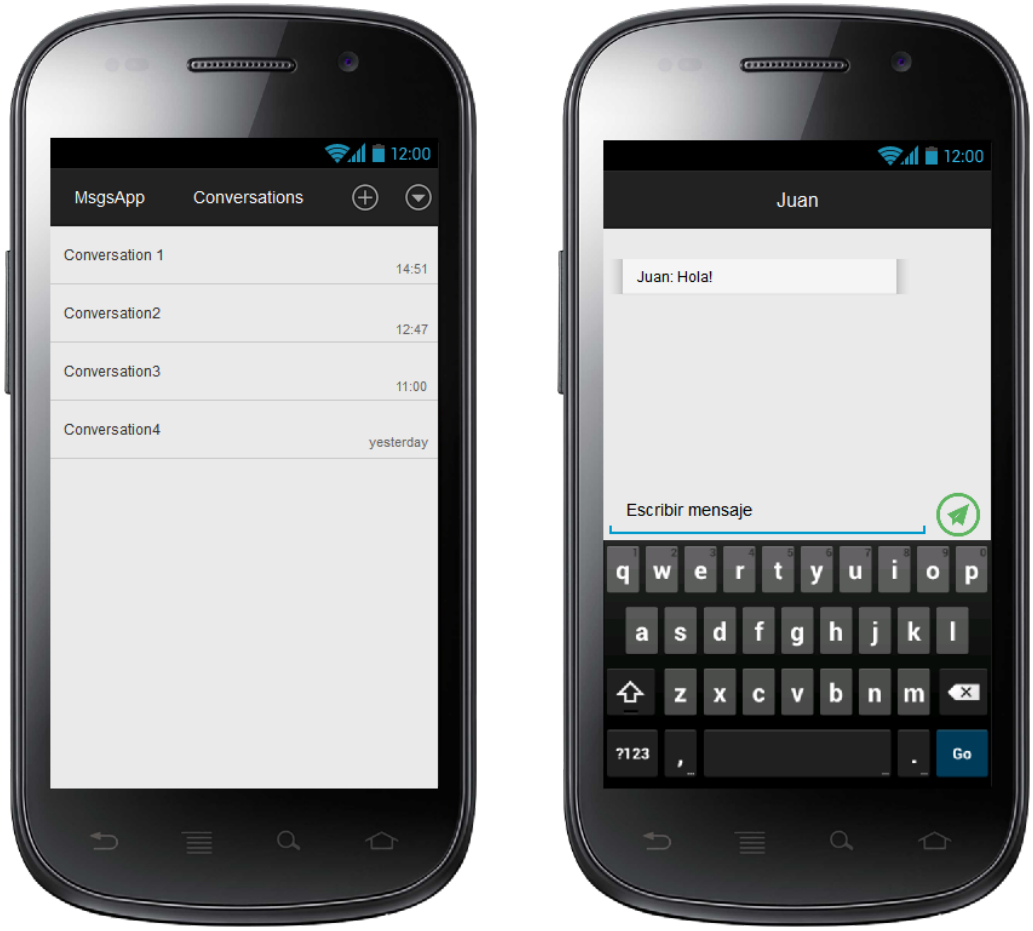
- Visualizar las conversaciones activas.
- Abrir cualquiera de las conversaciones activas para acceder al chat.
- Acceder al menú para crear un nuevo grupo.
- Acceder a la lista de usuarios para comenzar una nueva conversación.

La siguiente actividad con mayor importancia es la del chat. Se hace una distinción entre si la conversación es individual o grupal, ya que las funcionalidades son diferentes, por ejemplo, debe ser posible eliminar una conversación con cualquier otro usuario en cualquier momento, sin embargo, borrar un grupo sólo debería ser posible para el administrador. Así como que los grupos tienen la necesidad de poder añadir a usuarios al grupo, mientras que una conversación entre dos usuarios es precisamente eso, y por tanto añadir a otros usuarios no tiene cabida.

4.3.2. Diseño gráfico

En cuanto al diseño gráfico de la aplicación y como ya se ha comentado en el apartado anterior, será lo más parecida posible a interfaces a las que los usuarios ya están acostumbrados, facilitando el uso de la aplicación y haciéndolo lo más amigable e intuitivo posible.

A continuación se muestran dos maquetas de las actividades más importantes de la aplicación, la actividad principal o de *conversaciones* y la actividad de *chat*. Sin embargo, son maquetas que sólo pretenden hacer una idea del resultado final, pues éste será indudablemente diferente, aunque ciñéndose a los mismos principios.



(a) Diseño preliminar de la actividad *conversaciones* de la aplicación Android.

(b) Diseño preliminar de la actividad *chat* de la aplicación Android

Figura 4.3: Maquetas mostrando una primera aproximación al apartado gráfico de la aplicación Android

4.4. Requisitos técnicos

Para que todo funcione correctamente, los equipos sobre los que se ejecutará la aplicación deberán cumplir una serie de requisitos, los cuales se exponen a continuación.

4.4.1. Servidor

Para que el servidor funcione correctamente debe contar con las siguientes propiedades:

- **Python 2.7**. Charm, la biblioteca criptográfica que implementa las funcionalidades de la Encriptación Basada en la Identidad es una biblioteca para Python, que funciona tanto en su versión 2.7 como en su versión 3. Sin embargo, la compilación de Charm es distinta dependiendo de la versión de Python, y en este caso se ha elegido la 2.7.

- **Charm**. La biblioteca criptográfica que implementa las funcionadlidades de IBE es totalmente indispensable, y por lo tanto, también lo son sus dependencias [23]:
  - Pyparsing, una alternativa a al parsing de expresiones regulares. [24].
  - GMP (GNU Multiple Precision), una biblioteca para realizar operaciones aritméticas con gran velocidad y precisión.[25].
  - PBC Library (Pairing Based Cryptography), la biblioteca que se encarga del tipo de criptografía sobre el que está basada actualmente la Encriptación Basada en la Identidad [26].
  - OpenSSL, proyecto open source muy amplio que se encarga entre otras cosas de proporcionar herramientas para los protocolos SSL y TLS [27].
- **Flask**, un microframework muy ligero para hacer servidores en Python de forma sencilla [10].
- **MySQL**, la base de datos de la aplicación, donde se almacena toda la información acerca de los usuarios usa MySQL [28].
- Conexión a internet, como es obvio, el servidor no podría recibir ninguna petición. Sin embargo, las pruebas y la funcionalidad de la aplicación en general se puede observar en local.
- Sistema Operativo Linux. Éste último no es un requisito absoluto, tanto Charm, como Python y sus bibliotecas pueden funcionar en Windows o Mac, sin embargo, sólo se han realizado pruebas en un entorno Linux, por lo que es muy recomendable tener un sistema con un Sistema Operativo de este tipo.

4.4.2. Cliente Android

En el lado del Cliente no hay casi requisitos técnicos. Esto es esencial ya que el lado del Cliente debería tener el menor número de trabas posibles para poder utilizar la aplicación. En este caso, como el Cliente se relaciona con un servidor REST, sólo tiene que realizar peticiones HTTP de tipo POST a través de SSL. Por lo tanto, los únicos requerimientos por parte del Cliente son:

- Sistema Operativo Android con la versión 15 de la API.
- Conexión a internet para poder interactuar con el servidor REST.

Capítulo 5

Implementación

El propósito de este capítulo será detallar todo lo que tiene que ver con la implementación del proyecto. Se comenzará por un breve repaso de las características hardware y software del equipo en el que se ha desarrollado el proyecto, para proseguir con una pequeña explicación acerca de las plataformas más importantes que han sido utilizadas y cuál ha sido su función. A continuación se comentará la estructura del servidor, listando todos los ficheros que lo componen y explicando su función. De igual manera, se comentará la estructura del Cliente, analizando la funcionalidad específica de cada una de las clases. Más adelante, se hablará con mayor detalle acerca de la librería Charm, utilizada para implementar las partes de IBE y algunas características afines a la implementación de la aplicación. Por último se entrará en temas más específicos del desarrollo tanto del Servidor como del Cliente, explicando explicando con una mayor profundidad detalles más complejos de la implementación de ambos.

5.1. Equipo de desarrollo

A continuación se muestran las características del equipo en el que se ha desarrollado el proyecto separadas en hardware y software.

5.1.1. Hardware

El proyecto se ha llevado a cabo en un **Alienware 15** con las siguientes características:

- Procesador **Intel i7-4720HQ** con 4 núcleos físicos (8 virtuales) a **2.6GHz**.
- Memoria **RAM 16GB** DDR3
- Discos duros **82GB** SSD y **1TB** disco magnético.
- Tarjeta gráfica **Nvidia GTX980M**

5.1.2. Software

Sistemas Operativos

- **Windows 10 Home 64bits** como sistema operativo principal en el que se desarrolló todo lo relacioando con Android.
- Máquina virtual **VMware Workstation 12 Player** con **Ubuntu 14.04 LTS 64bits** para todo lo relacionado con el servidor.
- **Genymotion** con varias máquinas virtuales Android (APIs 16 y 18) para las pruebas relacionadas con Android.

Software de programación

- **Sublime Text 2** para realizar toda la codificación del servidor [29].
- **Android Studio 1.5.1** para el desarrollo de la aplicación Android [30].
- **Git** y **Bitbucket** para la gestión de versiones.
- **MySQL Workstation** y **MySQL Server** para la base de datos.

Software de edición

- **LaTeX** para la creación de este documento.
- **GIMP 2** para la edición de imágenes [31].
- **yEd** para la creación de diagramas [32].
- **Evolus Pencil** para la creación de maquetas para el diseño gráfico de la aplicación móvil [33].

5.2. Plataformas

En esta breve sección se listarán las tecnologías más importantes que se han utilizado para desarrollar el proyecto y en qué partes del mismo se ven implicadas y cómo.

5.2.1. Python

Python es el lenguaje de programación empleado para realizar toda la codificación de los componentes que forman el Servidor. Es un lenguaje de programación muy directo, en el que se consigue mucho en pocas líneas de código y que destaca principalmente por la amplitud, variedad y facilidad de uso de sus bibliotecas, las cuales han provisto soluciones sencillas a una gran variedad de aspectos de la implementación. Algunas a destacar son **MySQL Connector** para las funciones de conexión y ejecución de sentencias SQL, o **json**, para tratar con ficheros JSON con gran facilidad. Por no hablar de sus funciones criptográficas para las funciones de Hash o cifrado simétrico.

5.2.2. Flask

Framework para Python utilizado para crear servidores ligeros. Fácil y directo de utilizar, fue de gran utilidad para la creación de los tres componentes que forman el Servidor, proporcionando además, una manera muy sencilla de comunicación a través de SSL.

5.2.3. OpenSSL

Herramienta utilizada para la creación de los certificados empleados para la comunicación segura mediante SSL entre todos los componentes de la aplicación.

5.2.4. Android SDK

Por supuesto, para desarrollar la aplicación para dispositivos Android se necesita el Android SDK utilizando el lenguaje de programación Java.

5.3. Biblioteca Charm

La biblioteca Charm para Python fue la elegida para proporcionar la Encriptación Basada en la Identidad (IBE). La idea original era explorar las distintas implementaciones y esquemas que la biblioteca provee y elegir uno para llevar a cabo la criptografía del proyecto. Sin embargo, estando Charm todavía en desarrollo se descubrieron las siguientes limitaciones:

1. La mayoría de las implementaciones no permitían cifrar mensajes, sólo pares de números.

2. No había ningún esquema que permitiese tanto cifrar/descifrar como firmar/verificar.

Tras asimilar los puntos anteriores, se decidió que no quedaba más remedio que emplear dos esquemas diferentes, uno para la cifrado y otro para la firma. Así pues, la PKG tiene dos claves públicas maestras y dos claves privadas maestras, y los usuarios deben generar dos claves privadas diferentes. Esta no es la situación ideal, pero es la solución que se dio al problema encontrado.

Finalmente, los esquemas seleccionados fueron *HybridIBEnc* [34] para la cifrado y descifrado de mensajes y *Hess* [35] para la firma y verificación. Se ha de añadir además, que contrario a lo comentado hasta ahora sobre IBC, el modelo empleado para firmar, *Hess*, requiere además de la generación de la clave pública de usuario. Dicha clave se genera a partir del nombre de usuario correspondiente y es accesible para cualquier usuario autenticado.

5.4. Estructura del Servidor

En esta sección se cubrirá la funcionalidad básica de cada componente del servidor, explicando punto por punto su respectiva implementación, cubriendo toda la funcionalidad de cada componente, pero sin entrar en demasiado detalle. Ciertas partes del código que se consideran esenciales para el proyecto sí que serán explicadas en profundidad. Estas partes se cubrirán en la sección 5.6 para el Servidor y la sección 5.7 para el Cliente. Antes de comenzar, es importante comentar que todas las respuestas dadas por cualquiera de los Servidores que componen la parte del Servidor de esta aplicación dan sus respuestas en formato JSON siguiendo el mismo patrón, de esta forma se facilita mucho la comunicación y favorece la consistencia de la implementación.

5.4.1. PKG

Como ya se comentó en la parte de análisis (Sección 3) y sobre todo en la parte de diseño (Sección 4), la PKG es el componente encargado de generar las claves IBE, tanto las maestras, como las privadas de cada usuario, por lo que su funcionalidad se centrará en precisamente eso: generar claves. A continuación se muestran los distintos apartados que son accesibles para el Servidor de PKG y a los que llama directamente el Servidor de Base de Datos.

InitialSetup

Es una función de la PKG que se encarga de generar las claves maestras si nunca se han generado antes. No se llama directamente desde el Servidor de Base de Datos, sino que es llamada por Login (función que se verá a continuación) si detecta que nunca se han generado las claves maestras. Una vez llamada, genera las claves maestras pública y privada, tanto del esquema criptográfico IBE necesario para cifrar, como del necesario para firmar (ver sección 5.3). Una vez generadas, las pasa a formato JSON y las cifra con AES antes de guardarlas en un fichero. La clave usada para la cifrado es una derivación de la contraseña del administrador y una sal.

Login

Es la función que se llama cuando el administrador se identifica con su nombre de usuario y contraseña al intentar iniciar el servidor. Si la contraseña llega a la PKG significa que ya ha sido contrastada con la base de datos, y que por lo tanto es correcta. Luego se utiliza la contraseña y la sal anterior para descifrar el fichero con las claves maestras generadas y el Servidor PKG se pone en marcha tras guardarlas en memoria.

GetPks

Es una función que podría llamar potencialmente cualquier usuario, ya que sólo devuelve las claves públicas maestras, tanto la del esquema para cifrar/descifrar como del esquema para firmar/verificar. Sin embargo, para mantener el orden y la modularidad, en esta arquitectura sólo el Servidor de Base de Datos podrá hacer esta petición.

GenKeysById

Es la función más importante de la PKG y la que le da el nombre, puesto que es la que se encarga de generar las claves privadas de los usuarios a partir de su identidad, en nuestro caso, su nombre de usuario. Esta cadena es única para cada usuario, como ya veremos en la sección de la Base de Datos (Sección 5.6.1).

RegenerateKeys

Es una función necesaria para el componente PKG como tal, pero las implicaciones que tiene llamar a esta función están fuera del alcance de este proyecto. RegenerateKeys, está implementada, y sencillamente vuelve a generar las claves maestras, cargándolas en memoria. Sin embargo, volver a generar las claves maestras inutiliza todas las claves de los usuarios, ya que sin la antigua clave pública maestra, sus claves no valen de nada. Por lo tanto, habría que volver a generar claves para todos los usuarios. Y eso no es todo, el servidor de la base de datos debería obtener uno a uno todos los mensajes almacenados en la base de datos, descifrarlos con la clave antigua, cifrarlos con la nueva y volverlos a introducir en la base de datos. Siendo éste un caso un poco excesivo para el alcance de la aplicación, es una función inutilizada.

5.4.2. DBServer

El Servidor de Base de Datos, que será referido como DBServer para acortar, es el Servidor que acumula más funcionalidades de los tres. No sólo es el intermediario para todas las funcionalidades de la PKG sino que gestiona la conexión con la base de datos, y por lo tanto todas las peticiones relacionadas de algún modo con el almacenamiento de información. Es además el Servidor al que se conecta el Administrador para iniciar sesión e iniciar propiamente la aplicación. Se puede decir que DBServer ocupa la posición de Servidor de Aplicaciones, dando respuesta a la mayor parte de los casos de uso del Cliente. A continuación se expondrán una por una todas sus funcionalidades.

Funcionalidad de comunicación con la base de datos

Tal y como indica el nombre, DBServer gestiona la comunicación con la base de datos, a continuación se verá dónde y cómo. Por lo demás, se explicará la base de datos en un mayor detalle en las secciones 5.4.4 y 5.6.1.

Funcionalidades de intermediario

Una vez más, se resalta la función de DBServer como intermediario entre el resto de componentes y la PKG. Para ser más específicos, desde DBServer se accede a las funciones de Login, GetPks y GenKeysById de la PKG.

Login

DBServer es un Servidor que no guarda sesiones, y el motivo por el que no lo hace es el siguiente. Debido a las consideraciones de seguridad, el servidor no almacena las contraseñas de los usuarios en claro, al mismo tiempo, las claves IBE generadas para los usuarios se almacenan en el servidor cifradas a partir de la propia contraseña de los usuarios, como ya se verá en detalle en la sección 5.6.3. Debido a esto, el servidor no puede acceder a dichas claves, y por lo tanto no puede transmitirlas al Cliente, o en este caso al Servidor que se encarga de cifrar y descifrar mensajes. Para poder acceder a estas claves, DBServer necesita la contraseña, por lo que el usuario debe volver a introducirla cada vez que se requiera acceder a sus claves, es decir, cada vez que escribe o recibe un mensaje. Por lo tanto, ya que DBServer tiene que estar pidiendo la contraseña muy frecuentemente, se decidió hacer el Servidor Stateless y requerir el nombre de usuario y contraseña en todas las peticiones para asegurar que el Cliente está registrado.

Login es la funcionalidad del servidor que comprueba que el usuario esté registrado en la base de datos. Para ello, realiza el Hash de la contraseña y comprueba que sea igual al que está almacenada.

SignUp

SignUp es la funcionalidad de DBServer que permite a un usuario registrarse. Para ello, comprueba que el nombre de usuario no exista previamente, ya que al ser el sistema criptográfico basado en la identidad utilizando el nombre de usuario para generar las claves, dos usuarios no pueden poseer el mismo nombre de usuario. Una vez comprobado que el nombre de usuario está libre, procede a almacenar la contraseña tras pasarla por la función Hash. Además, almacena una sal aleatoria en la base de datos para utilizar en la cifrado de otros ficheros.

GenKeys

Es una de las funciones que llaman a una funcionalidad de PKG, en este caso GenKeysById, sin embargo, es digna de mención en DBServer pues complementa su funcionalidad. En un principio, el Cliente podría cambiar de clave cuando quisiese, y todavía puede, sin embargo, la criptografía se hace totalmente transparente para el usuario por lo que no se le deja generar un par de claves cuando quiera, sino que la aplicación lo hace una única vez de manera automática cuando se registra el usuario.

GetKeyHistory

Existe un fichero JSON asociado a cada usuario con su historial de claves, que se va expandiendo según se generan más claves. No obstante, como se acaba de explicar, el cliente sólo genera claves una vez. Esta es una funcionalidad pensada para devolver el fichero entero ideada para un Cliente que podría llegar a hacer uso de él. Por lo que utiliza la contraseña del Cliente para descifrarlo y lo envía a través de SSL al Servidor. En la implementación actual, esta función no es llamada en ningún momento.

NewMsg

Es una función que sencillamente almacena un nuevo mensaje en la base de datos. Sin embargo, devuelve también el identificador del mensaje, información esencial para que el Cliente sepa que su mensaje ha sido enviado con éxito al servidor.

NewMsgGroup

Es una función muy parecida a NewMsg, sin embargo, en vez de recibir el identificador de otro usuario, recibe el identificador de un grupo y almacena el mensaje en la base de datos con la información pertinente.

NewNotification

Aunque ya se verá la estructura de la base de datos más detalladamente en futuras secciones, conviene explicar brevemente lo que es una notificación. A diferencia de un mensaje, que se almacena una única vez por mensaje enviado, una notificación se crea por usuario al que va destinado el mensaje y posee la información necesaria para que pueda descifrar el mensaje. Explicado esto, NewNotification, es simplemente la función que guarda la nueva notificación en la base de datos.

NewGroup

Es la función que se encarga de añadir una entrada en la base de datos cada vez que se crea un grupo. Sin embargo, hay un campo variable en todo grupo que no es adecuado para una base de datos relacional. Esto es la lista de participantes. Debido a que su inclusión en la base de datos hubiera provocado la creación de una nueva tabla en la que además, habría muchos datos repetidos, se optó por almacenar la lista de participantes en un fichero JSON aparte. De este modo, NewGroup no sólo crea una entrada en la tabla de grupos de la base de datos, sino que además crea el fichero JSON correspondiente. Este fichero no está cifrado, al igual que los participantes del grupo habrían ido en claro en la base de datos. Sin embargo, implementar el cifrado del fichero sería trivial con la infraestructura ya creada, reutilizando el mismo sistema que con el fichero que gestiona el historial

de claves de cada usuario. Por lo que si en algún momento se llegase a considerar que la lista de participantes de cada grupo debiese permanecer protegida, podría implementarse rápidamente.

AddParticipantToGroup

Como se ha explicado en la función anterior, la lista de grupos está definida en un fichero JSON para cada grupo. AddParticipantToGroup simplemente modifica este fichero para añadir un nuevo nombre a la lista.

GetGroupInfo

Es una función que devuelve un JSON con cierta información acerca de un grupo. Tras asegurarse de que el usuario que la pide forma parte del grupo, proporciona información como la fecha de creación, la lista de participantes y quién es el administrador del grupo.

DeleteGroup

Función que comprueba en primer lugar si el usuario que ha pedido borrar un grupo es el administrador de éste y luego procede a eliminar la entrada correspondiente de la base de datos y eliminar el fichero con la lista de participantes.

GetUnreadNotifications

Una de las funciones más importantes del servidor, busca todos los mensajes marcados como no leídos del usuario y los devuelve en formato JSON, con toda la información que necesita alguien con las claves apropiadas para descifrar el mensaje.

ConfirmMsgRead

Función que marca una notificación en la base de datos como leída. De tal forma, el mensaje dejará de aparecer cuando el Cliente llame a GetUnreadNotifications.

5.4.3. Server

El nombre de Server se eligió para este componente en base a que es el único componente con el que se relaciona directamente el Cliente, y por lo tanto, es el único que existe desde su punto de vista. Además de su obvia función como intermediario entre el Cliente y el resto de Servidores, se encarga de cifrar y descifrar los mensajes para el Cliente. Debido a esto, es el Servidor con mayor carga computacional. A continuación se detallará su funcionalidad punto por punto.

Funcionalidad de intermediario

Como se ha venido diciendo, Server hace de intermediario entre el Cliente y DBServer, por lo tanto recibe todas sus peticiones. No se comentarán estas funciones puesto que ya se han detallado antes y lo único que hace Server es crear una nueva petición a partir de la del Cliente y redirigirla a DBServer.

NewMsg

Es la función que se llama cuando un usuario envía un mensaje a otro usuario en una conversación no grupal. Es por tanto la función que se encarga de cifrar el mensaje como es debido, utilizando tanto cifrado simétrico (AES) como asimétrico (IBE). Además, esta función coopera con DBServer llamando tanto a NewMsg como a NewNotification. También necesita obtener las claves pertinentes para cifrar los mensajes, pero todos los detalles acerca de la cifrado y descifrado de mensajes se podrán encontrar en la sección 5.6.2.

NewMsgToGroup

Función muy parecida a NewMsg pero en vez de enviar un mensaje a un único usuario en una conversación privada, NewMsgToGroup está hecha para enviar mensajes a todos los usuarios de una conversación grupal. De este modo, llama una vez a NewMsg de DBServer y tantas veces a NewNotification como usuarios haya en el grupo. Para ello, también debe llamar a GetGroupInfo para obtener la lista de participantes del grupo.

GetNewNotifications

Es la función que llama a GetUnreadNotifications para obtener todos los mensajes no leídos de un usuario. Esta función obtiene de la notificación toda la información que necesita para descifrar los mensajes. Pedirá las claves pertinentes a DBServer y retransmitirá los mensajes descifrados al Cliente.

5.4.4. DatabaseFunctions

DatabaseFunctions es el fichero de utilidad que contiene todas las funciones de conexión con la base de datos. Como ya se ha visto antes, se trata de una base de datos MySQL, y la biblioteca que se utiliza para gestionarla es mysql.connector. No se entrará en detalles en esta sección debido a que las funciones de conexión con la base de datos no tienen demasiado interés, sencillamente cumplen con su función. En la Sección 5.6.1 se verá el esquema de la base de datos, las funciones de este fichero sencillamente proveen una manera de insertar y retirar la información necesaria en la base de datos.

5.4.5. Util

Util es un fichero que incluye, como su nombre indica, ciertas utilidades que son necesarias para el funcionamiento de la aplicación y que se pueden dividir en criptográficas y control de entrada.

Criptografía

Todas las funciones criptográficas que se emplean en el proyecto están definidas en este fichero, así como otras funciones necesarias para llevar a cabo algunas de ellas.

AES Entre las funciones de AES, se encuentran cifrar y descifrar. Para implementarlas se utiliza la biblioteca Crypto.Cipher de Python. Cabe destacar que el vector de inicialización es aleatorio y que las claves de sesión que se utilizan para cifrar cada mensaje son también aleatorias. Además, como AES necesita bloques de tamaño múltiplo de 16, también se incluyen las funciones de *pad* y *unpad* que añaden bits aleatorios para mantener el tamaño de los bloques.

Se incluye también la función que genera claves a partir de la contraseña del usuario y una sal. Para ello se utiliza la función PBKDF2. Por último, también se incluye una función para generar bits aleatorios que sirvan de sal.

IBE Como se explicó en la Sección 5.3, la biblioteca Charm no está completamente desarrollada, y una de sus limitaciones forzó el uso de dos esquemas diferentes, uno para cifrar y otro para firmar. En este fichero, se incluyen las funciones necesarias para cifrar/descifrar con un esquema y para firmar/verificar con el otro. Además, se incluyen dos funciones de vital importancia, que son las de serializar y deserializar, las cuales permiten transportar fácilmente los objetos criptográficos fruto de IBE a través de la red.

Control de entrada

Las funciones de control de entrada están pensadas para evitar que el usuario pueda introducir caracteres extraños que puedan dañar el sistema, evitando, entre otras cosas, SQL Injection. Para esto, la función analiza la entrada y sólo permite cadenas con caracteres seguros, buscando, por ejemplo, espacios, puntos y comas, asteriscos o paréntesis.

Además incluye una función para asegurarse de que las contraseñas introducidas son seguras, asegurándose de que contienen más de ocho caracteres, mayúsculas, minúsculas, números y símbolos permitidos.

5.5. Estructura del Cliente Android

En esta sección se procederá a comentar la estructura de clases de la aplicación Android, comenzando por las actividades que definen las diferentes "pantallas"de la aplicación, dando un breve repaso por los adaptadores para mostrar la información obtenida del Servidor, continuando con la explicación de la conexión con el servidor y finalizando con algunas clases de utilidad para una mejor implementación.

5.5.1. Actividades

Las actividades son los componentes principales de una aplicación Android, a continuación se explica la funcionalidad de cada actividad y con qué otras actividades se comunican y cómo.

LoginActivity

Debido a que el Servidor requiere que la aplicación proporcione tanto el nombre de usuario como la contraseña en todas las peticiones, la actividad que se muestra al iniciar la aplicación es precisamente la que permite el inicio de sesión. LoginActivity es una actividad que muestra un panel de inicio de sesión clásico, y que permite tanto inciar sesión como registrar a un nuevo usuario. Sólo cuando una de las dos acciones se ha completado satisfactoriamente se muestra la siguiente actividad.

ConversationsActivity

Se podría decir que es la actividad principal de la aplicación ya que muestra una lista de todas las conversaciones activas señalando las que tienen mensajes no leídos, mostrándolas primero. Además, permite la creación de grupos y de conversaciones individuales. La manera de almacenar las conversaciones individuales y grupales en el dispositivo es mediante ficheros JSON. Estos ficheros se guardan en la memoria interna del teléfono y contienen toda la información necesaria, como la fecha de creación, los mensajes, cada uno con autor, fecha y si está o no leído, participantes del grupo en el caso de conversaciones grupales, etc. Desde esta actividad se puede acceder a la mayoría de las actividades que se explicarán a continuación.

ChatActivity

Si ConversationsActivity es la actividad principal, ChatActivity junto a GroupChatActivity forman las actividades más importantes de la aplicación, y es que son las que permiten escribir y visualizar mensajes, cumpliendo el requisito básico de una aplicación de mensajería. Desde esta actividad se puede acceder a ConversationInfoActivity.

GroupChatActivity

Es una actividad idéntica a la anterior, pero para conversaciones grupales, por lo que emplea una función de envío de mensajes distinta y accede a la actividad GroupInfoActivity.

NewGroupActivity

Su propósito principal es el de crear un nuevo grupo. Es una actividad sin demasiado contenido y que se beneficiaría mucho de ciertas mejoras que están fuera del alcance de este proyecto, como añadir participantes durante la creación del grupo, o elegir más de un administrador.

UserListActivity

Se encarga de mostrar una lista con todos los usuarios registrados, bien para empezar una conversación individiual con uno de ellos si se accede desde ConversationsActivity, o para añadir a uno de ellos como participante de un grupo. Es una actividad incompleta, que existe tal y como es para poder probar una de las funcionalidades de la aplicación, ya que mostrar todos los usuarios registrados es impensable si la lista fuese muy larga, y por otra parte, proporcionar a cada usuario con la lista total de los nombres de los otros usuarios no es una buena política de privacidad.

ConversationInfoActivity

Actividad que muestra la información sobre una conversación y permite borrarla por completo.

GroupInfoActivity

Actividad que muestra la información de una conversación grupal, permite visualizar la lista de participantes, y borrar el grupo si se es el administrador de dicho grupo.

PreferencesActivity

Es una actividad que no tiene una vista como tal, ya que no se han considerado preferencias para la aplicación, sin embargo, proporciona métodos estáticos para gestionar los ficheros de preferencias de forma global, haciendo la aplicación más mantenible.

5.5.2. Adaptadores

En la aplicación móvil, es necesario en varias actividades mostrar una lista, ya sean conversaciones, mensajes o usuarios. Para mostrar dichas listas, se ha empleado el widget *RecyclerView* y para mostrar la información, *CardView*. Con la combinación de ambos se consigue una apariencia moderna y una presentación clara de la información. Para transformar la información en la visualización concreta que presenta la aplicación son necesarios los adaptadores. A continuación se explicarán, sin entrar en mucho detalle, los distintos adaptadores y su función.

ChatRecyclerAdapter

Es el adaptador que se encarga de mostrar los mensajes en los chats, se le pasa una lista con mensajes y los muestra incluyendo autor, fecha y mensaje. Colorea en verde los mensajes del usuario. Es utilizada tanto por ChatActivity como por GroupChatActivity.

ConversationsRecyclerAdapter

Se encarga de mostrar la lista de conversaciones activas, muestra el nombre del otro usuario o del grupo, y el último mensaje, incluyendo su autor y fecha. Es utilizada por ConversationsActivity.

UserListRecyclerAdapter

Muestra la lista de usuarios, por lo que sólo muestra el nombre de usuario. Utilizada por UserListActivity.

5.5.3. Conexión con el servidor

Como ya se ha comentado en anteriores apartados, toda la comunicación entre componentes se lleva a cabo mediante peticiones HTTP, por lo tanto se necesitaba utilizar una biblioteca de este tipo. Es por esto que la comunicación con el servidor se realiza mediante la biblioteca Volley que aporta mucha flexibilidad a la comunicación.

ServerInterface

Es la clase que gestiona la conexión con el servidor de manera directa, conociendo todas las URLs y generando las peticiones para las distintas funcionalidades de las que se ha hablado en la Sección 5.4.

Server

Es la clase que abstrae al Servidor para el Cliente, incluye todas las llamadas a las funciones de ServerInterface pero haciendo automático, entre otras cosas, introducir como parámetros el nombre de usuario y contraseña en cada petición.

Callbacks

Los callbacks son funciones llamadas cuando el servidor responde a una petición. Volley emploea este tipo de funciones, y se desarrollan normalmente en las clases en las que se espera recibir la petición. Por ejemplo, en LoginActivity se implementa el callback para procesar la respuesta del servidor a una petición de inicio de sesión. Por lo tanto, estas funciones están repartidas a lo largo de toda la aplicación, desarrollándose allí donde son más necesarias.

HttpsTrustManager

Como se explicado anteriormente, la comunicación entre componentes se realiza siempre cifrada mediante SSL. Sin embargo, generar certificados oficiales está fuera del alcance de este proyecto, por lo que se trabaja con certificados auto-firmados. Debido a esto no se pueden verificar, por lo que se necesita crear un gestor de confianza propio para aceptar esta clase de certificados. Sin embargo, y para hacer las cosas más sencillas, se ha decidido aceptar todos los certificados sin verificarlos. Se es perfectamente consciente de que no verificar ningún certificado es prácticamente igual a no utilizar SSL, sin embargo, si el proyecto llegase a llevarse a cabo con un fin más que académico, los certificados serían generados apropiadamente y por tanto verificables, y este problema no existiría. HttpsTrustManager es una clase que provee una función que permite ignorar la verificación de los certificados.

CustomRequest

Volley tiene algunos problemas con sus peticiones POST, y en ocasiones no es capaz de reconocer los parámetros que se envían en el cuerpo de la petición. CustomRequest es una clase que toma las medidas necesarias para que este error no tome lugar, y poder realizar las peticiones sin mayor problema. Como todas las respuestas del Servidor vienen en formato JSON, CustomRequest realiza peticiones de este tipo.

5.5.4. Útiles

Los útiles son clases de apoyo que facilitan ciertas tareas necesarias para el Cliente. A continuación se explicarán algunas de las más importantes.

JsonHelper

La aplicación Cliente también necesita almacenar información en memoria, como ya se ha explicado antes, los mensajes y las conversaciones activas se guardan en ficheros para que puedan visualizarse los mensajes ya recibidos sin necesitar conexión a internet. Para almacenar estos ficheros y modificarlos, se ha creado la clase JsonHelper, que cuenta con todas las funcionalidades para tratar automáticamente el almacenamiento y retirada de información de estos ficheros. Esta funcionalidad se verá con más detalle en la Sección 5.7.2.

NotificationChecker

NotificationChecker es una clase que existe para preguntarle al servidor si hay mensajes nuevos, y en caso de que los haya, avisar a la clase adecuada para que se actualice como corresponda. En el caso de ConversationsActivity, deberá poner la conversación con nuevos mensajes la primera de la lista y avisar de que hay mensajes nuevos, en el caso de ChatActivity o GroupChatActivity, deberá mostrar los mensajes nuevos. Los detalles específicos del funcionamiento de esta clase se verán en la Sección 5.7.1.

Clases de apoyo

Las clases de apoyo son clases cuyo único propósito es facilitar el trabajo de otras clases, sobre todo de los adaptadores. Son clases que contienen la información necesaria para mostrarse en un determinado momento, como por ejemplo, ChatMessage, que tiene los campos necesarios para mostrar en un mensaje, que son: el autor, la fecha, el mensaje, y si ha sido leído o no. De esta forma se puede pasar una lista a los adaptadores para que la muestren. Las clases de apoyo no se explicarán en detalle ya que carecen de interés más allá de su función general.

5.6. Servidor - Codificación detallada de problemas específicos

Aunque ya se ha explicado la estructura general del trabajo, hay partes de la implementación que es mejor ver en profundidad, para comprenderlas del todo, y para asegurar que cumplen los requisitos. A continuación se explican algunas de estas partes de la implementación, en concreto, del servidor.

5.6.1. Base de datos

Se ha hablado muy a menudo de la base de datos de la aplicación, sin embargo, no se ha visto cómo ha sido implementada. A continuación se verán sus diferentes tablas y campos, justificando las decisiones de diseño y explicando detalles de importancia.

Users La tabla Users es la que se encarga de almacenar a los usuarios de la aplicación. Guarda su **nombre de usuario**, su **contraseña**, su **clave pública**, una **sal**, y el **rol** del usuario en la aplicación.

En cuanto al nombre de usuario, como ya se ha dicho antes, debe ser único, por lo que es la clave primaria de esta tabla. La contraseña se almacena tras pasarla por la función Hash **sha256** con una sal que la propia biblioteca *passlib* para Python se encarga de introducir. El campo de clave pública es para la clave pública del esquema *Hess*, que aunque no es necesario, sirve para indicar si se han generado las claves para el usuario y hace que recuperar la clave sea más rápido que descifrar el fichero de historial de claves del usuario. Sólo se incluye la clave pública del esquema criptográfico utilizado para firmar, porque para el esquema *HybridIBEnc* basta con la pública maestra y el identificador. En cuanto a la sal, se genera para utilizarla al cifrar el fichero del historial de claves de cada usuario. El rol es una clave foránea a la clave primaria de la siguiente tabla.

Roles La tabla Roles es una tabla que cumple un propósito muy limitado en este momento, pero que sin duda sería necesaria de ser ampliada la aplicación. Sólo contiene una columna, el **rol**, que puede ser o *Admin* o *User*, con la única diferencia que sólo un usuario registrado como *Admin* puede iniciar el servidor. En un futuro, de haber distintos tipos de usuarios con diferentes permisos, en esta tabla podrían estar aparte de los roles, dichos permisos.

Messages Es la tabla que se encarga de guardar los mensajes que son enviados por cada usuario. Contiene el **id** de cada mensaje como clave primaria, el **autor** del mensaje, el propio **mensaje**, y dos campos que no pueden estar completos a la vez, el **usuario de destino**, y el **grupo de destino**, en función de si el mensaje se envía a un único usuario o a un grupo.

Notifications A diferencia de la tabla de mensajes, que guarda una entrada por mensaje de cada emisor, la tabla de notificaciones guarda una entrada por mensaje hacia un destinatario. Por lo que para ahorrar espacio, sólo tienen la referencia al identificador del mensaje. Así se evita repetir el mismo mensaje al enviar a grupos, en los que varios usuarios reciben el mismo.

Los campos son el citado **id** del mensaje, el **usario de destino**, la **posición de la clave** en el mensaje y su **longitud**, la longitud del **hash**, la **posición del mensaje**, la **fecha** de emisión y si el usuario ha marcado el mensaje como **leído**. Los campos de longitud y posición son necesarios para descifrar el mensaje, y serán completamente explicados en la siguiente sección.

Groups La tabla que se encarga de almacenar los diferentes grupos creados por los usuarios. Contiene el **id** del grupo como clave primaria, el nombre del **administrador** del grupo y el **nombre** del grupo. Como ya se comentó antes, los participantes del grupo están en un fichero aparte cuyo nombre está basado en el identificador de cada grupo.

5.6.2. Cifrado y descifrado de mensajes

Uno de los objetivos más importantes de este proyecto, si no el más importante, era el de establecer un esquema de autenticación basado en la identidad. Por lo tanto, es de gran importancia explicar cómo se ha llevado a cabo la cifrado de los mensajes. A continuación se explicará de manera detallada dicho proceso. Se recomienda ver la Figura 5.1, en la que se simula que *Alice* envía un mensaje a un grupo en el que están *Bob* y *Carol*, para comprender mejor la estructura del mensaje.

1. El usuario escribe un mensaje y éste es enviado al Servidor, conjunto con información como a qué usuario o grupo está dirigido y quién es el emisor.

2. Una vez en posesión del mensaje, el Servidor, genera una clave de sesión aleatoria y cifra el mensaje con AES.

3. Para la cifrado asimétrica, el servidor pide a la PKG las claves públicas maestras.

4. El servidor, o bien conoce quién es el destinatario, o el grupo de destino. En el primer caso, cifra la clave con la clave pública del destinatario (la cuál no necesita, ya que el esquema de cifrado sólo necesita la pública maestra y la identidad de destino para cifrar) y concatena la clave cifrada, poniéndola al principio del mensaje. En el caso de que el destinatario sea un grupo, primero debe pedir a DBServer la lista de participantes de dicho grupo, y luego cifrar la clave una vez por usuario, poniéndolo siempre al principio del mensaje.

5. Preparándose para firmar, el Servidor pide a DBServer que mire en el historial de claves del usuario emisor, y le de la clave privada del esquema de firma.

6. Teniendo el contenido del mensaje listo, se realiza un Hash de todo el mensaje y se firma con la clave privada del emisor. Este Hash también se concatena al principio del mensaje.

7. El Servidor hace una petición para almacenar un nuevo mensaje en la base de datos.

8. Por último, el Servidor hace tantas peticiones como usuarios de destino haya, creando notificaciones en las que incorpora para cada usuario, en qué posición del mensaje pueden encontrar la clave cifrada con su clave pública.

A la hora de recibir el mensaje, el proceso sería parecido, pero a la inversa.

1. En una de sus peticiones regulares, el Cliente vería que tiene mensajes sin leer.

2. El Servidor pediría las notificaciones correspondientes, y de ellas obtendría el mensaje cifrado, y toda la información necesaria para encontrar la clave cifrada y el Hash.

3. Para poder descifrar, el Servidor pediría la clave pública del usuario emisor, así como las claves públicas maestras. También se obtendrían las claves privadas del usuario para descifrar la clave del mensaje.

4. Una vez en su poder, quitaría el Hash del mensaje, realizaría su propio Hash y verificaría, tras descifrar el hash con la clave pública del usuario emisor que su Hash y el enviado son el mismo. Así se asegura tanto que el mensaje no ha sido modificado, como que el emisor es quien dice ser.

5. Una vez asegurado que el mensaje no ha sido alterado, se procede a localizar la clave del mensaje, cifrada con la pública del receptor.

6. Una vez obtenida la clave, se procede a descifrar el mensaje.

7. Finalmente, este mensaje es enviado al Cliente.

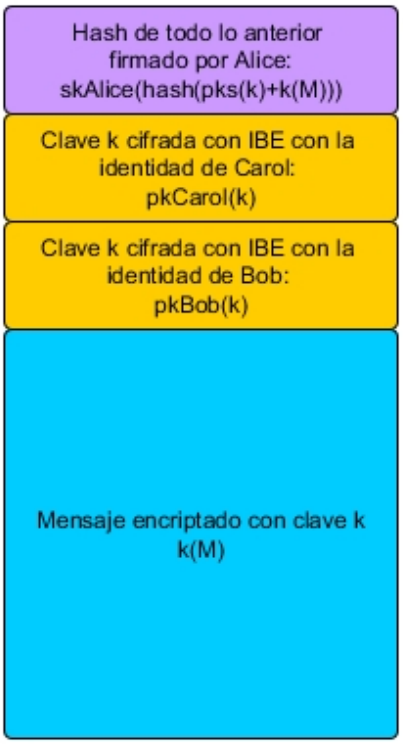


Figura 5.1: Estructura del mensaje almacenado en DBServer. Ejemplo concreto en el que *Alice* envía un mensaje a *Bob* y a *Carol*

5.6.3. Almacenamiento de claves

Como se ha explicado anteriormente, aunque es posible, sólo se permite a los usuarios generar un par de claves, sin embargo, el Servidor permite generar cuantas se quieran, y almacena el historial de claves en ficheros JSON, de los cuales se recuperan cuando es necesario. Sin embargo, almacenar las claves es algo delicado, ya que con ellas podrían descifrarse los mensajes. Es por ello que el historial de claves se guarda cifrado, y para hacerlo todavía más seguro se utiliza una derivación de la contraseña de cada usuario para cifrar cada fichero. De esta forma, cada fichero de cada usuario está cifrado con una clave distinta fruto de la derivación de la contraseña de dicho usuario y una sal aleatoria que se genera al registrarse. La función de derivación empleada es PBKDF2.

5.7. Cliente - Codificación detallada de problemas específicos

Aunque la mayoría de problemas difíciles de resolver están en la parte del Servidor, el Cliente tiene algunos detalles de la implementación en los que se requiere una mayor profundidad, los más importantes son cómo se reciben las nuevas notificaciones, y cómo se gestionan las conversaciones en la memoria interna del teléfono.

5.7.1. Recepción de notificaciones

Para la recepción de notificaciones se consideraron dos diseños: o bien el servidor recordaba a cada Cliente cada cierto tiempo que tenían mensajes sin leer, o bien el Cliente preguntaba cada cierto tiempo si tenía mensajes sin leer. Se optó por la segunda opción para evitar sobrecargar al servidor enviando mensajes a todos los usuarios, cuando algunos de ellos podrían tener la aplicación inactiva.

Para llevar a cabo esta funcionalidad en la parte del Cliente, se utilizaron *Alarmas*, de tal manera que una clase específica del Cliente llamada *NotificationChecker* preguntase al servidor cada cierto tiempo (específicamente siete segundos) si había mensajes nuevos y actuase en consecuencia. De esta manera, si en efecto hay mensajes nuevos, la propia clase se encarga de añadirlos a los ficheros donde se almacenan las conversaciones (ver próxima sección), y de indicar al Servidor que los mensajes han sido leídos para que dejen de aparecer en las futuras peticiones.

5.7.2. Almacenamiento en memoria interna

Con el propósito tanto de acelerar el cargado de mensajes en memoria, como de poder acceder a la información descargada sin conexión, se decidió almacenar las conversaciones en la memoria del teléfono. Para poder acceder fácilmente a las conversaciones, se generan dos tipos de ficheros, los de conversaciones individuales, y los de conversaciones grupales. Son ficheros en formato JSON muy parecidos, pues ambos contienen el listado de mensajes de la conversación, con fecha y autor, pero también tienen sus diferencias. Como son la lista de participantes, sólo presente en las conversaciones grupales, y qué usuario es el administrador del grupo, también únicamente presente en las conversaciones grupales.

La funcionalidad para trabajar con estos ficheros fue implementada en la clase **JsonHelper**, que ya ha sido descrita anteriormente. Los nombres de los ficheros están derivados directamente de o bien el nombre del usuario con el que se tiene la converación individual, o el id del grupo si la conversación es grupal. Sin embargo, para que al iniciarse la aplicación se sepa qué conversaciones hay, es necesario un fichero con nombre fijo que contenga los ficheros abiertos, este es el llamado fichero de **control** que contiene precisamente esa lista.

Capítulo 6

Pruebas

Las pruebas son un elemento necesario del desarrollo de cualquier proyecto para asegurar que el funcionamiento de la aplicación es el deseado. Por supuesto, esta aplicación no es diferente, y múltiples pruebas fueron realizadas para cada uno de los componentes de la aplicación. A continuación se explicarán los distintos tipos de pruebas realizadas tanto para el Servidor como para el Cliente, cómo se hicieron y por qué se hicieron cómo y cuándo se hicieron.

6.1. Servidor

El Servidor, o la combinación de los tres Servidores que lo forman, es el componente más complejo de la aplicación. Además, es el componente que incluye el mayor número de tecnologías nuevas para el autor, haciendo que la probabilidad de error sea aún mayor. Por eso mismo se siguió la estrategia de desarrollar componente a componente y no empezar a desarrollar el siguiente sin haber probado antes lo anterior. De este modo, se asegura una mayor calidad de código, y se aíslan mejor los problemas que van surgiendo. Las pruebas realizadas progresivamente en el servidor pueden dividirse en **unitarias** y de **integración**.

6.1.1. Pruebas unitarias

Las pruebas unitarias son pruebas para asegurar que una función determinada cumple su objetivo de manera aislada. Este tipo de pruebas son muy útiles para asegurarnos de que las funciones que se desarrollan funcionan debidamente, y fueron empleadas en todos los módulos del servidor cada vez que se terminaba un determinado set de funciones. Entre los grupos de pruebas unitarias más importantes destacan las siguientes:

- Funciones relacionadas con la base de datos. La base de datos fue creciendo según se iban desarrollando otros componentes del servidor, y por lo tanto, funciones para obtener, modificar e insertar datos en la base de datos fueron introduciéndose cuando era necesario. Para asegurarse de que dichas funciones llevaban a cabo su objetivo, se llamaba a las distintas funciones en batería con combinaciones tanto posibles o normales, como imposibles (por ejemplo intentar insertar dos usuarios con el mismo nombre). Comprobando que la información obtenida o insertada era correcta (a menudo empleando *MySQL Workbench*, se comprobaba también que las funciones eran correctas.

- Funciones criptográficas. Debido a la naturaleza del proyecto, existen múltiples funciones de índole criptográfica, tanto de tipo simétrico como asimétrico. Para asegurarse de que dichas funciones cumpliesen su objetivo se llevaron a cabo pruebas para comprobar entre otras cosas, que pudiesen cifrar y descifrar con las claves adecuadas, que no pudiesen hacerlo con las inadecuadas, que la firma y la verificación funcionasen, etc.

- Servidores Flask. Según se iban desarrollando los Servidores, se crearon interfaces HTML con formularios para comprobar en el navegador si la información era recogida y devuelta correctamente.

6.1.2. Pruebas de integración

Las pruebas de integración existen para asegurar que varios componentes independientes son capaces de llevar a cabo un trabajo de forma conjunta. Debido a que el Servidor está compuesto por tres Servidores individuales, las pruebas de integración son completamente necesarias. Las principales pruebas de integración realizadas son las siguientes:

- El último tipo de pruebas unitarias que se ha mencionado: probar el servidor mediante interfaces HTML desde el navegador, eran a menudo pruebas unitarias, sin embargo, todo depende de la funcionalidad que se vaya a probar. Desde el navegador también se hicieron múltiples pruebas de integración, probando funcionalidades como el registro de usuarios o la creación de grupos, que atañe tanto al Servidor, como a la base de datos. O las pruebas desde el Servidor de Base de Datos que incluían pedir claves a la PKG, por poner un ejemplo.
- En Server, se probaron todas las funcionalidades que se iban a llamar desde el dispositivo móvil haciendo una batería de llamadas a funciones e imprimiendo los resultados por pantalla para comprobar que el resultado era el deseado. De esta forma, como el servidor llamaría a esas mismas funciones cuando el Cliente hiciese la petición, se estaba asegurando el funcionamiento de estas funcionalidades en conjunto. Además, la mayoría de estas funciones requerían contactar con los otros Servidores para llevar a cabo su objetivo.

6.2. Cliente Android

En el caso del Cliente, realizar una batería de pruebas unitarias no tenía tanto sentido como en el servidor. Esto es debido a que la complejidad del Cliente es bastante inferior a la del servidor, y por lo tanto no había tanta necesidad de probar funcionalidades individuales sino el funcionamiento de la aplicación en conjunto.

Es por este motivo que el grueso de pruebas que se realizaron fueron llevadas a cabo de manera manual, con el emulador *Genymotion*, probando a mano que todo funcionase correctamente y que se cumpliesen los diferentes requisitos especificados en la Sección de análisis.

Sin embargo, también se realizaron pruebas automatizadas, para ello existe la clase TestActivity que sirvió para probar funcionalidades más específicas, como la correcta escritura y lectura de ficheros, o los primeros intentos de conexión con el Servidor .

Capítulo 7

Conclusiones y trabajos futuros

Uno de los objetivos del trabajo era el estudio de la Seguridad Basada en la Identidad como alternativa a métodos tradicionales de gestión de identidades. Para esto, IBC posee una serie de ventajas que la convierten en un candidato digno al considerar este problema, como son que el usuario no tenga que almacenar sus propias claves, la generación de claves bajo demanda, la delegación de capacidades criptográficas a otras entidades, la facilidad de revocación de claves, etc. Todas estas propiedades son requerimientos esenciales para responder a los principales retos (muchos de ellos relacionados con el tratamiento de la privacidad) que existen hoy en día en el ámbito TIC [36], [37], y en los cuales tienen cabida diferentes soluciones fundamentadas en IBC [9], [38].

Sin embargo, también posee una serie de desventajas que hacen que haya que elegir con cuidado para qué situación utilizar este modelo para la gestión de identidades. La PKG debe ser totalmente confiable, ya que tiene la capacidad de generar las claves de todos sus usuarios, introduciendo un problema de *key escrow*. Y es que como ya se ha visto, su órgano central, la PKG aporta el mayor número de las ventajas, pero también el mayor número de inconvenientes. El hecho de que una única entidad tenga acceso a tanta información crítica entraña un gran riesgo, pues si la seguridad de dicha entidad fuese violada, todo el sistema se vendría abajo. Este hecho reduce la posibilidad de implementar IBC a gran escala a organismos que puedan asegurar su seguridad, y puedan asegurar hacerse cargo de las consecuencias si llegase a haber una brecha en dicha seguridad. Por otra parte, a pequeña escala y especialmente relacionado con el Internet of Things, IBE puede llegar a ser muy útil para establecer comunicación segura y gestionar adecuadamente las identidades en entornos controlados [9]. Por otro lado, existen variantes de IBC que tratan de eliminar o reducir el impacto de la vulnerabilidad originada por el problema del *key escrow* [39].

El objetivo principal del TFG era el desarrollo de la aplicación que se ha descrito a lo largo del documento. Un Cliente de mensajería para Android y un Servidor que hace las funciones de Servidor de aplicaciones y PKG. Por lo tanto, el objetivo principal ha sido cumplido, aunque la imposibilidad de hacer que Charm funcionase en Android supuso que el Cliente no puede cifrar por sí mismo, el esquema de autenticación está implementado y los requisitos especificados cumplidos. Por otra parte, la aplicación es bastante compleja, por lo que es normal que pueda mejorarse en varios aspectos.

Para comenzar con los aspectos de seguridad, los certificados están auto-firmados. Para que esta aplicación fuese realmente segura, los certificados deberían ser auténticos, firmados por una Entidad Certificadora. No verificar la validez de los certificados es algo completamente impensable para una aplicación de mercado. El Servidor de la PKG sólo debe ser contactado por DBServer, sin embargo, esto no se asegura de ningún modo. Un cortafuegos adecuado asegurando que el Servidor de la PKG sólo respondiese a peticiones provenientes de DBServer ayudaría a hacer el sistema más seguro. Permitir al usuario cambiar de contraseña también contribuiría positivamente a la seguridad global de la implementación, que se podría fortalecer aún más mediante procedimientos de autenticación basados en múltiples factores [40].

En cuanto al Servidor, podría desplegarse en la red, en varios equipos, aprovechando la facilidad de distribución y mejorando el rendimiento. Se podría permitir el cambio de claves maestras de la PKG, realizando todos los cambios necesarios para que los clientes no notasen la diferencia.

Asimismo, es altamente recomendable el despliegue de una arquitectura mediada por una DMZ (*DeMilitarized Zone*) para preservar la seguridad perimetral de las redes en las que incluyan cada uno de los servidores.

Respecto del cliente, se podrían realizar muchas mejoras para mejorar la usabilidad y añadir funcionalidades muy útiles a la aplicación Android. Como por ejemplo permitir añadir participantes durante la creación de los grupos, eliminar participantes, transferir derechos de administrador de grupo, ofreciendo información más detallada sobre las conversaciones y grupos. Se debería poder buscar usuarios por nombre, en lugar de que apareciese toda la lista de usuarios registrados al crear una conversación o añadir participantes a un grupo.

Además, podría desarrollarse una interfaz gráfica para convertir el Servidor de cifrado en un cliente para ordenadores con un Sistema Operativo basado en Linux. Permitiendo usar el ordenador como Cliente y así emplear cifrado de extremo a extremo. Este cliente ya está desarrollado, por lo que sólo sería necesario implementar la interfaz gráfica.

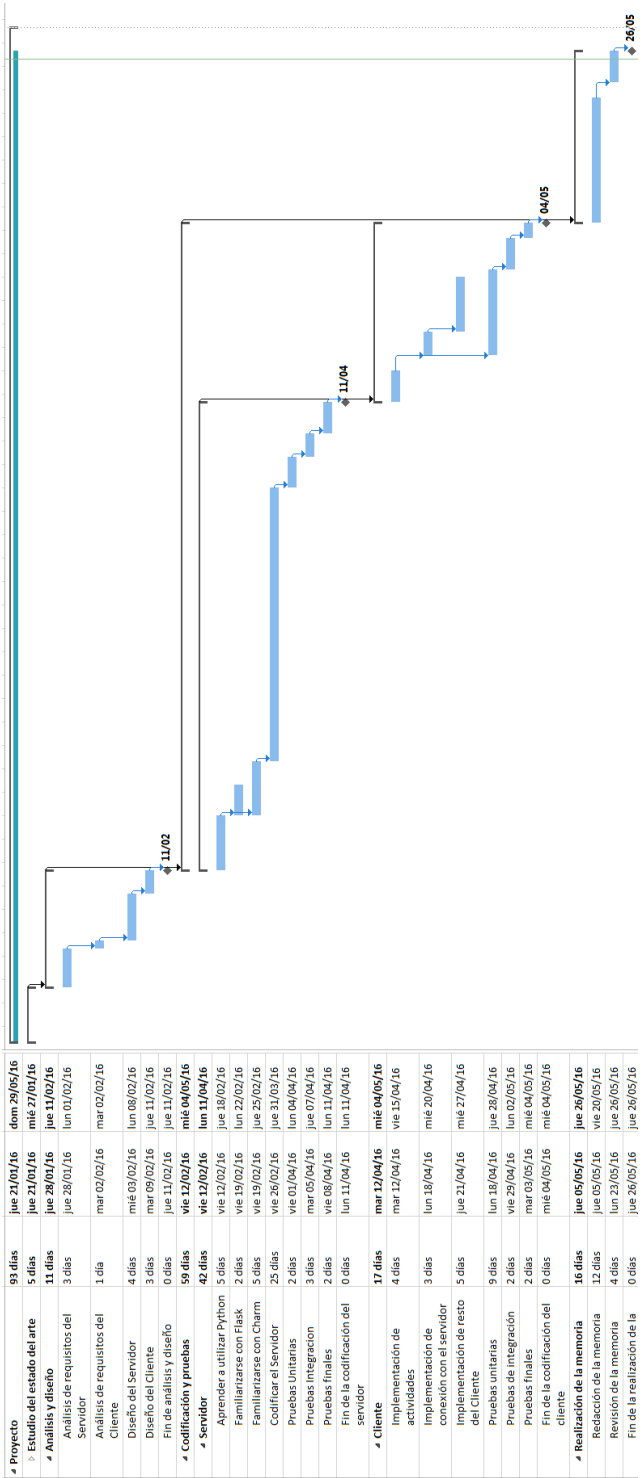
El último objetivo era de carácter personal y consistía en familiarizarse con las tecnologías usadas en el proyecto. Este objetivo ha sido cumplido con creces. El número de tecnologías diferentes empleadas, de las cuales muchas eran desconocidas para el autor de este TFG, ha sido alto, lo que ha repercutido tanto en la dificultad del proyecto como en el aprendizaje y formación necesaria para efectuarlo.

Para terminar, este proyecto constituye un ejemplo de uso de varias tecnologías, incluyendo la biblioteca no estandarizada y aún en desarrollo Charm, por lo que aporta valor de cara a futuras implementaciones al proporcionar un ejemplo de uso de la biblioteca, además de señalar varias de sus ventajas y posibles mejoras.

Anexo A

Planificación del proyecto

En este anexo se muestra un diagrama de Gantt con la planificación del proyecto.



Proyecto	93 días	jue 21/01/16	dom 29/05/16
Estudio del estado del arte	5 días	jue 21/01/16	mié 27/01/16
Análisis y diseño	11 días	jue 28/01/16	jue 11/02/16
Análisis de requisitos del Servidor	3 días	jue 28/01/16	lun 01/02/16
Análisis de requisitos del Cliente	1 día	mar 02/02/16	mar 02/02/16
Diseño del Servidor	4 días	mié 03/02/16	lun 08/02/16
Diseño del Cliente	3 días	mar 09/02/16	jue 11/02/16
Fin de análisis y diseño	0 días	jue 11/02/16	jue 11/02/16
Codificación y pruebas	59 días	vie 12/02/16	mié 04/05/16
Servidor	42 días	vie 12/02/16	lun 11/04/16
Aprender a utilizar Python	5 días	vie 12/02/16	jue 18/02/16
Familiarizarse con Flask	2 días	vie 19/02/16	lun 22/02/16
Familiarizarse con Charm	5 días	vie 19/02/16	jue 25/02/16
Codificar el Servidor	25 días	vie 26/02/16	jue 31/03/16
Pruebas Unitarias	2 días	vie 01/04/16	lun 04/04/16
Pruebas Integracion	3 días	mar 05/04/16	jue 07/04/16
Pruebas finales	2 días	vie 08/04/16	lun 11/04/16
Fin de la codificación del servidor	0 días	lun 11/04/16	lun 11/04/16
Cliente	17 días	mar 12/04/16	mié 04/05/16
Implementación de actividades	4 días	mar 12/04/16	vie 15/04/16
Implementación de conexión con el servidor	3 días	lun 18/04/16	mié 20/04/16
Implementación de resto del Cliente	5 días	jue 21/04/16	mié 27/04/16
Pruebas unitarias	9 días	lun 18/04/16	jue 28/04/16
Pruebas de integración	2 días	vie 29/04/16	lun 02/05/16
Pruebas finales	2 días	mar 03/05/16	mié 04/05/16
Fin de la codificación del cliente	0 días	mié 04/05/16	mié 04/05/16
Realización de la memoria	16 días	jue 05/05/16	jue 26/05/16
Redacción de la memoria	12 días	jue 05/05/16	vie 20/05/16
Revisión de la memoria	4 días	lun 23/05/16	jue 26/05/16
Fin de la realización de la	0 días	jue 26/05/16	jue 26/05/16

Figura A.1: Diagrama de Gantt con la planificación del proyecto

Anexo B

Ejemplos de casos de uso

Durante la parte de análisis se hizo referencia a ciertos casos de uso. A continuación se muestran dos de los más importantes para la aplicación, el envío de mensajes en el Cliente y el inicio del Servidor.

Nombre	**RF1 – El administrador inicia el servidor.**
Actores	Administrador.
Precondiciones	Ninguna.
Post-condiciones	El servidor está activo y esperando peticiones.
Descripción	El administrador pone en funcionamiento el servidor, para lo cual deberá identificarse.
Secuencia principal	**Paso** / **Acción** 1 / El administrador ejecuta el programa del servidor. 2 / El administrador introduce nombre de usuario y contraseña. 3 / El servidor comprueba que son correctas. 4 / El servidor se inicia y queda a la espera de peticiones.
Secuencia alternativa	**Paso** / **Acción** 3.1 / El nombre de usuario y contraseña introducidos son incorrectos. 3.2 / El servidor informa de este hecho y se detiene.

Figura B.1: Caso de uso del servidor: iniciar servidor.

Nombre	**RF9 – El usuario envía un mensaje.**
Actores	Usuario registrado.
Precondiciones	El usuario ha iniciado sesión. El usuario se encuentra visualizando sus conversaciones activas.
Post-condiciones	El mensaje se ha enviado y está almacenado en el servidor.
Descripción	El usuario envía un mensaje a otro usuario o grupo desde un chat.
Secuencia principal	**Paso** / **Acción** 1 / El usuario pulsa sobre una de sus conversaciones activas para acceder a ella. 2 / El usuario accede a la pantalla del chat donde pulsa en el recuadro para escribir. 3 / El usuario termina de escribir y pulsa el botón de enviar. 4 / El servidor recibe el mensaje y lo almacena en la base de datos.
Secuencia alternativa	N/A

Figura B.2: Caso de uso del cliente: enviar mensaje.

Anexo C

Primera arquitectura del Servidor

A continuación se muestra el primer diseño para la arquitectura del Servidor, que fue descartada al no ser posible cifrar en el lado del Cliente.

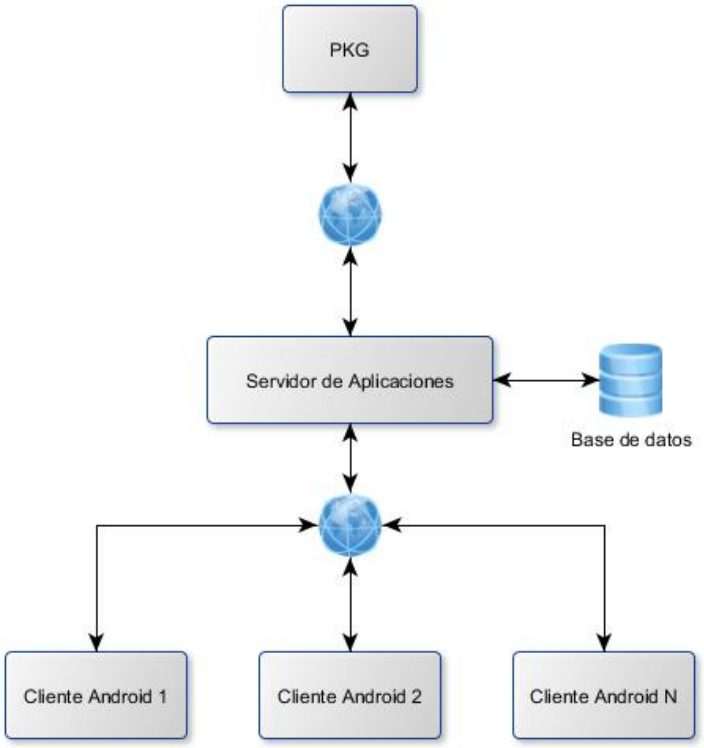


Figura C.1: Primer diagrama de arquitectura, con tres componentes principales, la PKG, el Servidor de Aplicaciones, y los clientes Android.

Anexo D

Escalabilidad del Servidor

Como ya se comentó en la parte de diseño, la arquitectura propuesta podría ser ampliada para mejorar la escalabilidad si la aplicación llegase a tener tantos usuarios que un único servidor estuviese desbordado con la pesada carga de cifrar y descifrar los mensajes de todos los usuarios. Para dar solución a este hipotético problema se proponía una solución en la que se multiplicaba el número de servidores hasta alcanzar un número adecuado al número de usuarios y las peticiones se distribuían a los diferentes servidores mediante un balanceador de carga. A continuación se muestra dicho esquema.

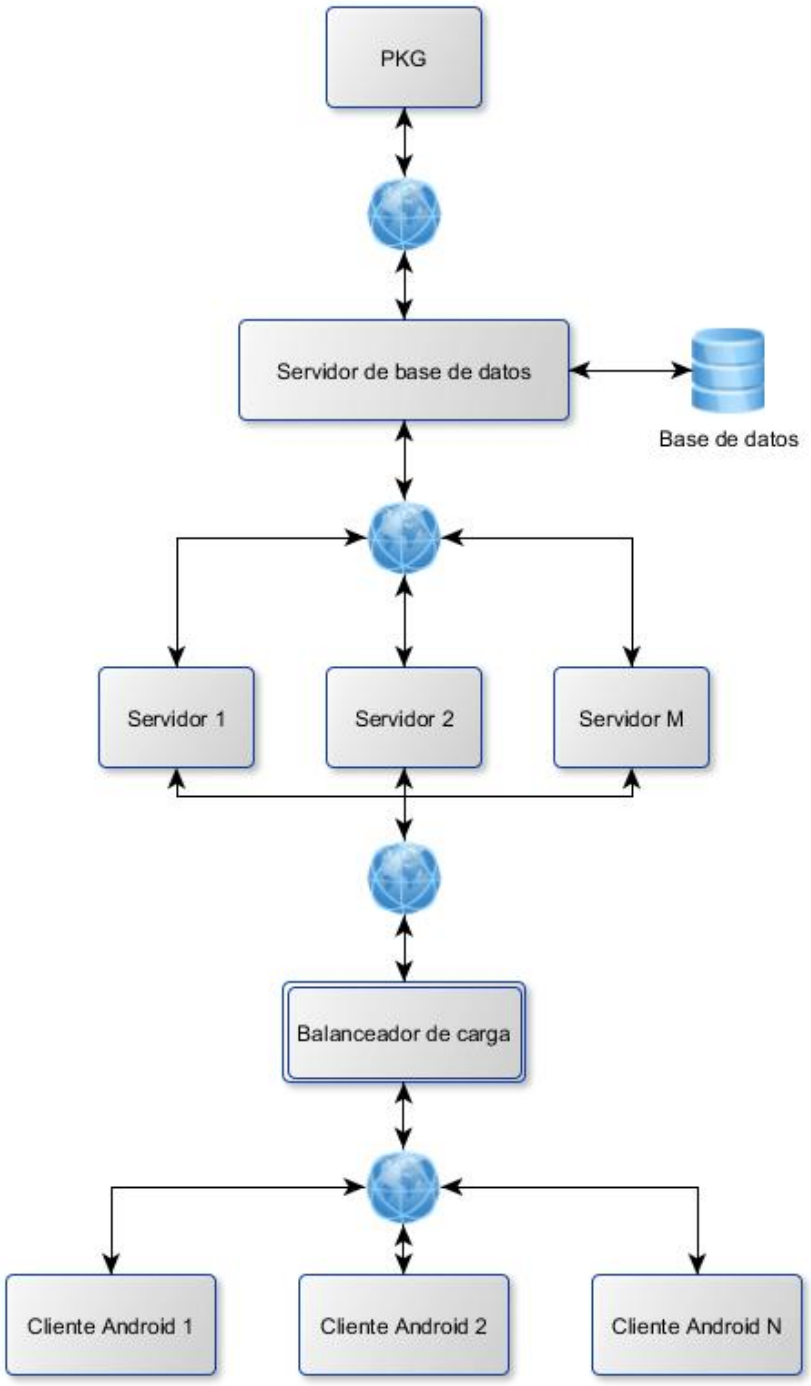


Figura D.1: Arquitectura propuesta para mejorar la distribución y escalabilidad del proyecto.

Anexo E

Generación de certificados

En el documento, se ha hablado de que toda comunicación a través de la red se ha llevado a cabo de manera segura a través de SSL, para llevar esto a cabo, es necesario generar unos certificados. Sin embargo, obtener los certificados de manera oficial está fuera del alcance del proyecto, por lo que se emplearon certificados auto-firmados. A continuación se muestra el proceso seguido para obtener los certificados utilizando la herramienta *OpenSSL* [27], [41].

1. Generar una clave privada RSA de 1024 bits de longitud.

   openssl genrsa -des3 -out server.key 1024

2. Generar un CSR

   openssl req -new -key server.key -out server.csr

3. Eliminar la contraseña de acceso a la clave.

   cp server.key server.key.org

   openssl rsa -in server.key.org -out server.key

4. Generar el certificado auto-firmado.

   openssl x509 -req -days 365 -in server.csr -signkey server.key -out server.crt

Bibliografía

[1] J. Gleick, *The Information: A History, a Theory, a Flood*. Pantheon Books, 2011, ISBN: 9780375423727 (vid. pág. 1).

[2] S. Singh, *The code book: The science of secrecy from ancient Egypt to quantum cryptography*. Anchor, 2011 (vid. pág. 1).

[3] B. Schneier, *Applied cryptography: Protocols, algorithms, and source code in C*. john wiley & sons, 2007 (vid. pág. 1).

[4] *The Identity-Based Encryption advantage*, https://www.voltage.com/resource/the-identity-based-encryption-advantage-a-proven-standard-for-protecting-information/, Accedido por última vez: 25-05-2016 (vid. págs. 1, 6, 8).

[5] R. Oppliger, *SSL and TLS: Theory and Practice*. Artech House, 2009 (vid. pág. 1).

[6] *Identity Based Encryption IBE 3.0 explained*, https://www.linkedin.com/pulse/identity-based-encryption-ibe-30-explained-martijn-kolenbrander, Accedido por última vez: 24-05-2016 (vid. págs. 1, 4, 8).

[7] *Charm*, http://jhuisi.github.io/charm/, Accedido por última vez: 17-05-2016 (vid. págs. 1, 14).

[8] L. Martin, *Introduction to identity-based encryption*. Artech house, 2008 (vid. pág. 1).

[9] D. J. Wu, A. Taly, A. Shankar y D. Boneh, «Privacy, discovery, and authentication for the internet of things», *ArXiv preprint arXiv:1604.06959*, 2016 (vid. págs. 1, 8, 39).

[10] *Flask*, http://flask.pocoo.org/, Accedido por última vez: 16-05-2016 (vid. págs. 1, 14, 21).

[11] A. Cavoukian y M. Dixon, *Privacy and Security by Design: An Enterprise Architecture Approach*. Ontario: Information y Privacy Commissioner, 2013 (vid. pág. 1).

[12] D. Shaw, L. Donnerhacke, R. Thayer, H. Finney y J. Callas, «OpenPGP message format», 2007 (vid. págs. 3, 6).

[13] *Internet x.509 public key infrastructure certificate and CRL profile*, https://www.ietf.org/rfc/rfc2459.txt, Accedido por última vez: 26-05-2016 (vid. págs. 3, 6).

[14] G. Á. Marañón, P. P. P. Garcıa y P. Bustamante, *Seguridad informática para la empresa y particulares*. McGraw-Hill, 2004 (vid. pág. 4).

[15] W. Diffie y M. E. Hellman, «New directions in cryptography», *Information Theory, IEEE Transactions on*, vol. 22, n.º 6, págs. 644-654, 1976 (vid. pág. 5).

[16] D. Boneh y M. Franklin, «Identity-based encryption from the weil pairing», en *Advances in Cryptology—CRYPTO 2001*, Springer, 2001, págs. 213-229 (vid. págs. 7, 8).

[17] S. S. Al-Riyami y K. G. Paterson, «Certificateless public key cryptography», en *Advances in cryptology-ASIACRYPT 2003*, Springer, 2003, págs. 452-473 (vid. pág. 8).

[18] *JPBC*, http://gas.dia.unisa.it/projects/jpbc/, Accedido por última vez: 17-05-2016 (vid. pág. 14).

[19] *Charm serialization API*, http://jhuisi.github.io/charm/developers.html, Accedido por última vez: 17-05-2016 (vid. pág. 14).

[20] *Spring*, https://spring.io/, Accedido por última vez: 17-05-2016 (vid. pág. 14).

[21] *Telegram*, https://telegram.org/, Accedido por última vez: 18-05-2016 (vid. pág. 19).

[22] *Whatsapp*, https://www.whatsapp.com/?l=es, Accedido por última vez: 18-05-2016 (vid. pág. 19).

[23] *Platform install manual*, http://jhuisi.github.io/charm/install_source.html, Accedido por última vez: 16-05-2016 (vid. pág. 21).

[24] *Pyparsing wiki home*, http://pyparsing.wikispaces.com/, Accedido por última vez: 16-05-2016 (vid. pág. 21).

[25] *The GNU Multiple Precision Arithmetic Library*, https://gmplib.org/, Accedido por última vez: 16-05-2016 (vid. pág. 21).

[26] *PBC library*, https://crypto.stanford.edu/pbc/news.html, Accedido por última vez: 16-05-2016 (vid. pág. 21).

[27] *OpenSSL*, https://www.openssl.org/, Accedido por última vez: 16-05-2016 (vid. págs. 21, 49).

[28] *MySQL*, https://www.mysql.com/, Accedido por última vez: 16-05-2016 (vid. pág. 21).

[29] *Sublime Text 2*, http://www.sublimetext.com/2, Accedido por última vez: 19-05-2016 (vid. pág. 24).

[30] *Android studio*, https://developer.android.com/studio/intro/index.html, Accedido por última vez: 19-05-2016 (vid. pág. 24).

[31] *Gimp*, https://www.gimp.org/, Accedido por última vez: 18-05-2016 (vid. pág. 24).

[32] *yEd*, https://www.yworks.com/products/yed, Accedido por última vez: 18-05-2016 (vid. pág. 24).

[33] *Evolus pencil*, http://pencil.evolus.vn/, Accedido por última vez: 18-05-2016 (vid. pág. 24).

[34] *Ejemplos de cifrado IBE de charm*, http://jhuisi.github.io/charm/_modules/ibenc_test.html, Accedido por última vez: 22-05-2016 (vid. pág. 25).

[35] *Ejemplos de firma IBE de charm*, http://jhuisi.github.io/charm/_modules/pksig_test.html, Accedido por última vez: 22-05-2016 (vid. pág. 25).

[36] D. Arroyo, J. Diaz y V. Gayoso, «International joint conference: Cisis'15 and iceute'15», en, Á. Herrero, B. Baruque, J. Sedano, H. Quintián y E. Corchado, eds. Cham: Springer International Publishing, 2015, cap. On the Difficult Tradeoff Between Security and Privacy: Challenges for the Management of Digital Identities, págs. 455-462, ISBN: 978-3-319-19713-5. DOI: 10.1007/978-3-319-19713-5_39. dirección: http://dx.doi.org/10.1007/978-3-319-19713-5_39 (vid. pág. 39).

[37] J. Diaz, S. G. Choi, D. Arroyo, A. D. Keromytis, F. B. Rodriguez y M. Yung, «Data privacy management, and security assurance: 10th international workshop, dpm 2015, and 4th international workshop, qasa 2015, vienna, austria, september 21-22, 2015. revised selected papers», en, J. Garcia-Alfaro, G. Navarro-Arribas, A. Aldini, F. Martinelli y N. Suri, eds. Cham: Springer International Publishing, 2016, cap. Privacy Threats in E-Shopping (Position Paper), págs. 217-225, ISBN: 978-3-319-29883-2. DOI: 10.1007/978-3-319-29883-2_14. dirección: http://dx.doi.org/10.1007/978-3-319-29883-2_14 (vid. pág. 39).

[38] K. R. Butler, S. Ryu, P. Traynor y P. D. McDaniel, «Leveraging identity-based cryptography for node id assignment in structured p2p systems», *Parallel and Distributed Systems, IEEE Transactions on*, vol. 20, n.º 12, págs. 1803-1815, 2009 (vid. pág. 39).

[39] X. Huang, Y. Mu, W. Susilo, D. S. Wong y W. Wu, «Certificateless signatures: New schemes and security models», *The Computer Journal*, vol. 55, n.º 4, págs. 457-474, 2012 (vid. pág. 39).

[40] F. Stajano, «Security issues in ubiquitous computing», en *Handbook of Ambient Intelligence and Smart Environments*, Springer, 2010, págs. 281-314 (vid. pág. 39).

[41] *SSL for flask local development*, http://kracekumar.com/post/54437887454/ssl-for-flask-local-development, Accedido por última vez: 24-05-2016 (vid. pág. 49).